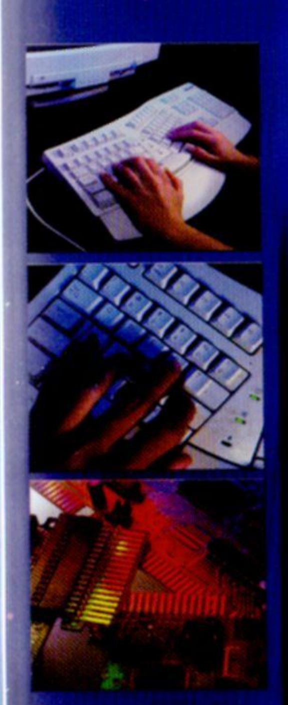
L. BABA-HAMED S. HOCINE

ALGORITHMIQUE ET STRUCTURES DE DONNEES STATIQUES

Cours et exercices avec solutions



Office des Publications Universitaires

L.BABA-HAMED
S. HOCINE
Université d'Oran Es-Senia
Faculté des Sciences

ALGORITHMIQUE ET STRUCTURES DE DONNÉES STATIQUES

Cours et Exercices avec Solutions

MODULE D'INFORMATIQUE TC SETI

Réimpression 2006



OFFICE DES PUBLICATIONS UNIVERSITAIRES
1, Place centrale de Ben-Aknoun (Alger)

L.BABA-HAMED
S. HOCINE
Université d'Oran Es-Senia
Faculté des Sciences

ALGORITHMIQUE ET STRUCTURES DE DONNÉES STATIQUES

Cours et Exercices avec Solutions

MODULE D'INFORMATIQUE TC SETI

Réimpression 2006



OFFICE DES PUBLICATIONS UNIVERSITAIRES
1, Place centrale de Ben-Aknoun (Alger)

EDITION: 2.08.4494
I.S.B.N : 9961.0.0630.5
Dépôt légal : 1270/2003

Avant propos

La notion d'algorithme est très générale et dépasse le cadre de l'informatique. En effet, nous utilisons tous des méthodes *algorithmiques, plus ou moins* consciemment, dans notre vie courante. Une recette de cuisine, le mode d'emploi d'un appareil sont des algorithmes : ils décrivent la suite des opérations à effectuer pour obtenir un résultat donné.

Signalons que le mot algorithme vient du nom du mathématicien arabe Abou Djafar Mohamed Ibn Moussa (IXe siècle), surnommé El Khawarizmi, et qui est l'un des fondateurs de l'arithmétique moderne.

En informatique, trouver un algorithme à un problème donné, ne se limite pas à écrire une suite d'opérations, de façon, à aboutir au résultat. On doit bien analyser le problème et raisonner, pour écrire un algorithme juste par conception, et non par des essais. On doit, également, construire des algorithmes, clairs, lisibles, précis, bien structurés et faciles à maintenir.

Ce polycopié, écrit à l'intention des débutants en informatique, et en particulier aux étudiants du Tronc-Commun SETI est largement inspiré des enseignements que nous dispensons au département du Tronc-Commun SETI de la faculté des sciences de l'université d'Oran ES_SENIA.

Nous tentons, à travers ce polycopié, de faire prendre conscience aux étudiants, de manière progressive, des notions fondamentales de l'algorithmique, des structures de données statiques, et de la décomposition d'un problème en tâches (procédures et fonctions) simples et faciles à résoudre.

Le lecteur trouvera, dans ce polycopié, une partie cours et une partie consacrée aux corrigés des exercices proposés à la fin de chaque

chapitre. Un grand nombre d'exercices ont été testé en travaux dirigés, et certains ont fait l'objet d'épreuves de contrôle, les autres exercices ont été inspirés d'ouvrages dont les références sont citées en bibliographie.

Dans notre enseignement de l'*algorithmique*, pour résoudre un problème, nous avons donné plus d'importance à l'énoncé qu'au langage de programmation. Nous exprimons nos algorithmes, dans un langage proche du français, simple, structuré, clarifié par des commentaires, et loin des contraintes d'un langage de programmation. Pour l'élaboration d'un algorithme, nous dégageons, de l'énoncé d'un problème posé, les tâches à accomplir et les objets à manipuler. Le passage, à la phase de programmation ne présente, généralement, pas de difficultés si l'algorithme est bien conçu ; c'est la raison pour laquelle nous n'en parlons pas dans notre polycopié.

Nous avons scindé notre polycopié en six chapitres. Le premier chapitre définit quelques notions informatiques, présente le langage algorithmique utilisé, et décrit les objets manipulés par un algorithme. Le second chapitre est consacré aux différents types élémentaires que l'on peut rencontrer. Les expressions et les instructions de base font l'objet du chapitre 3. Quant aux instructions structurées, elles sont abordées au chapitre 4. Le chapitre 5 présente les types de données structurés statiques, et le chapitre 6 définit les procédures et les fonctions. Enfin, nous terminons par le corrigé des exercices proposés à la fin de chaque chapitre.

Nous espérons que ce polycopié réponde aux attentes des étudiants, et qu'il puisse constituer une première étape de ce vaste domaine des temps modernes qu'est l'informatique.

Nos plus grands remerciements s'adressent à tous ceux et celles qui ont contribué à la formation des étudiants au Tronc-Commun.

Nous assumons l'entière responsabilité concernant les solutions proposées. C'est pourquoi nous prions les lecteurs de nous faire part de leurs remarques et de leurs critiques.

L. BABA-HAMED
S. HOCINE
babahl@yahoo.fr
soraya.hocine@univ-oran.dz

TABLE DES MATIERES

CHAPITRE IV - LES INSTRUCTIONS STRUCTUREES

CHAPITRE V - LES TYPES DE DONNEES STRUCTURES STATIQUES

CHAPITRE VI - PROCEDURES ET FONCTIONS

CHAPITRE I

NOTIONS PRELIMINAIRES

Ce chapitre constitue une introduction générale à l'informatique. Nous définissons, dans un premier temps, ce qu'est l'informatique, et présentons la structure générale d'un ordinateur ainsi que son fonctionnement. Les notions d'algorithme sont abordées, dans un second temps.

1 Initiation à l'informatique

Nous présentons dans ce qui suit la définition générale du mot **informatique**, et de l'outil informatique (ordinateur) avec ses composants.

1.1. Qu'est-ce que l'informatique?

L'informatique est la science du traitement automatique de **l'information**. Cette dernière représente un moyen de communication. Nous distinguons deux types d'informations; d'une part les instructions (directives spécifiant les actions élémentaires à exécuter) du programme que la machine devra exécuter, d'autre part, les données (souvent appelées opérandes) sur lesquelles la machine effectuera les traitements dictés par les instructions

1.2 L'ordinateur et sa structure

Pour mettre en application les développements théoriques qu'elle produit, l'informatique fait appel à ***l'ordinateur***, *qui est une machine* électronique utilisée pour le traitement automatique de l'information.
Tout *ordinateur est composé* essentiellement d'un processeur central, d'une mémoire centrale, d'une unité d'entrée, d'une unité de sortie et d'une mémoire auxiliaire.
Le **Processeur central** constitue l'élément essentiel de l'ordinateur (c'est le superviseur). Il exécute les opérations (arithmétiques,

logiques, et de comparaison), gère l'exécution des programmes, contrôle, et commande le fonctionnement des autres unités de l'ordinateur .

La **mémoire centrale** permet de mémoriser les programmes, les données en entrée, les résultats intermédiaires et les résultats finaux. Elle est constituée d'un ensemble de cases désignées chacune par une adresse. Le système de numération utilisé dans la mémoire est le système de numération binaire donc toutes les informations qui viennent s'y installer sont écrites (ou codées) en binaire.

Les symboles utilisés pour représenter les états de la mémoire sont les deux *chiffres* 0 et 1, chacun correspond à un **bit** (abréviation de binary digit) ; c'est la plus petite quantité d'information utilisable et stockable. Toutes *les cases* mémoires sont constituées d'un nombre fixe de bits. Selon le type d'un ordinateur, une case est constituée de 8 bits (octet en français ou un byte en anglais) ou d'un nombre plus grand de bits : 16, 24, 32,... et on parle d'un mot mémoire. La capacité de la mémoire centrale est limitée et est mesurée en kilooctets (un kilooctet = 2^{10} octets = 1024 octets).

L'**unité d'entrée** permet d'introduire, dans l'ordinateur, les informations que l'on désire traiter (exemple : le clavier, la souris, etc.).

L'**unité de sortie** permet, à l'ordinateur, de communiquer des informations au monde extérieur (exemple : l'écran de visualisation et l'imprimante).

La **mémoire auxiliaire** ne fait pas partie de l'ordinateur. Elle sert à archiver (stocker) des informations. A l'inverse de la mémoire centrale, la mémoire auxiliaire est une mémoire permanente (c'est à dire que son contenu n'est pas perdu en cas de coupure électrique), à accès lent et à capacité non limitée. Nous pouvons citer, par exemple, le disque dur, les disquettes, les CDs, etc.

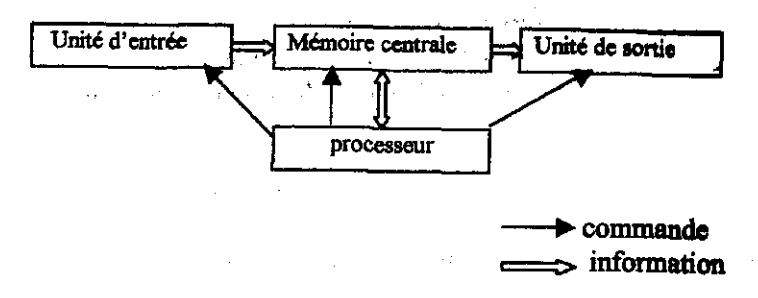


Schéma général d'un ordinateur

2. Notions d'algorithme

Nous définissons dans cette section l'algorithme, ses propriétés, le langage dans lequel il est exprimé ainsi que l'ensemble des objets qu'il manipule.

2.1 Définition et propriétés d'un algorithme

Le traitement que l'ordinateur effectue sur les données est spécifié par ce que l'on appelle un **algorithme**. L'algorithme décrit une suite finie, organisée et non ambiguë d'opérations élémentaires pour l'obtention d'une solution à un problème posé. Cette solution n'est pas unique en général. L'élaboration de tout algorithme nécessite une étape essentielle de réflexion à une méthode de résolution du problème posé, cette dernière constitue la phase d'**analyse** du problème. Elle permet, en outre, de mettre en évidence les données du problème (objets *supposés connus au départ*) et les résultats (objets attendus en sortie) . Pour son exécution sur *ordinateur*, un algorithme doit être transformé en un **programme** c'est-à-dire exprimé dans un langage de programmation (dit langage évolué) comme Pascal par exemple.

2.2 Exemple d'algorithme

L'algorithme qui fait la somme de deux *nombres entiers* A *et* B et affiche le résultat C peut être décrit comme suit :

introduire le premier nombre A
introduire le deuxième nombre B
faire l'addition A+B et mettre le résultat dans C
afficher C

Cet algorithme nécessite les données A, B et fournit le résultat C.

2.3 Définition du langage algorithmique (LA)

Tout algorithme est exprimé dans un langage proche du langage naturel appelé **langage algorithmique** ou **pseudo-langage**. Il décrit de manière complète et claire, les objets manipulés par l'algorithme *ainsi que l'ensemble des instructions* à exécuter sur ces objets pour obtenir des résultats. Il permet une bonne structuration d'un algorithme et aussi l'utilisation de *commentaires*. Ces *derniers sont* destinés à faciliter la compréhension d'un algorithme par un lecteur, mais ils sont ignorés, par le processeur, au moment de l'exécution. Ils sont encadrés par le symbole %.

2.4 Notion d'objet

Tous les **objets** qui rentrent en jeu pour l'exécution d'un algorithme doivent être déclarés avant toute utilisation : ceci constitue la partie déclaration d'un algorithme. Un objet peut être décrit par **un nom, un type, une valeur** et **une nature**. Le nom de l'objet sert à le désigner dans l'algorithme. Il doit être significatif, alphanumérique (mélange de lettres et de chiffres), commençant par une lettre et admettant le caractère _ (qui permet de fabriquer une chaîne d'un seul tenant, à partir de plusieurs mots), par exemple : PRIX, CODE_BANCAIRE, N1, X sont des noms corrects. Le type de l'objet caractérise l'ensemble des valeurs permises pour cet objet et les opérations qui lui sont autorisées (*cette notion sera détaillée dans le* chapitre II). La nature d'un objet, est sa caractéristique **constante** ou **variable** pour une donnée (nous verrons d'autres natures d'objets dans la suite de ce cours). Un objet est de nature variable, si sa valeur peut changer pendant l'exécution des actions de l'algorithme. Il est de nature constante si sa valeur est invariable. Nous pouvons voir une

variable comme une place, en mémoire de l'ordinateur, portant un nom.

Exemple

Les informations A, B et C, de l'exemple du paragraphe 2.2, représentent des objets de nature variable. La constante PI qui est égale à 3.14 (qui sert pour le calcul du périmètre ou surface d'un cercle) est un objet de nature *constante*.

En utilisant le LA, un objet de nature constante suit la déclaration :

Constante
Nom-de-constante = valeur-du-type

Où *nom-de-constante* et *valeur-du-type* représentent respectivement le nom et la valeur de la constante du type considéré.

Un objet de nature variable suit la déclaration:

Variable
Nom-de-variable : type

Où *nom-de-variable* et *type* représentent respectivement le nom et le type de la variable. Notons que le LA utilise des mots-clés, que le concepteur ne peut pas utiliser pour désigner ses objets, comme par exemple les mots constante et variable (nous découvrirons d'autres mots-clés dans la suite du cours). Pour les différencier des noms d'objets choisis par le concepteur, nous les soulignons.

Remarque

Si plusieurs variables sont de même type, *nom-de-variable* est remplacé par une suite de noms de variables séparés par une virgule.

Par exemple les objets A, B, C et PI sont déclarés comme suit :

<u>Constante</u>

PI = 3.14 % PI est une constante de type réel %

<u>Variable</u>

A, B, C: entier % A, B, et C sont des variables de type entier %

3. Exercices

1- On suppose que la capacité de la mémoire d'un ordinateur est de 1048576 mots et que la longueur du mot est de 16 bits. Donner la capacité de cette mémoire en octets et en *kilo-octets*.

2- Soient deux droites D1 et D2. Il s'agit de déterminer le point d'intersection, des deux droites, s'il existe.

a. Décrire les objets constituant la partie déclaration de ce problème.
b. Décrire la suite d'actions qui permet de déterminer le point d'intersection.

3- On dispose d'un sac non transparent, rempli de boules de différentes couleurs. On désire compter le nombre de boules rouges contenues dans le sac.

a. Décrire les objets constituant la partie déclaration de ce traitement.
b. Décrire la suite d'actions permettant de résoudre ce problème.

4- Pour encaisser, les clients se mettent en file, devant un guichet. Chaque client est muni d'une pièce d'identité et d'un chèque dûment rempli.

a. Décrire la partie déclaration de ce problème.
b. Décrire la suite d'actions qui permet à tous les clients d'encaisser.

CHAPITRE II

TYPES ELEMENTAIRES

Les types élémentaires sont des types simples, à valeur unique. On distingue les types standards : entier, réel, booléen, caractère et chaîne, et les types non standards : énuméré et intervalle. Un type est caractérisé par: un *nom* pour le désigner, un *domaine* (l'ensemble des valeurs permises), la *représentation des valeurs*, et *l'ensemble des opérateurs autorisés sur les objets du type*. Nous présentons, dans ce qui suit, les types standards, les types non standards et enfin la notion de compatibilité des types.

1. Les types standards

Les types standards sont les types : entier, réel, booléen, caractère et chaîne.

1.1 Le type entier

Nom : il est désigné par *entier*.

Domaine : est un intervalle fermé de l'ensemble des entiers relatifs, borné par les valeurs *min_entier* (plus petite valeur entière) et *max_entier* (plus grande valeur entière).

Représentation des valeurs: suit l'écriture des entiers relatifs (par exemple : 2, +12, -20) .

Opérateurs du type entier

- Les opérateurs arithmétiques : +, -, * et / (qui représente la division réelle)
- la division entière notée par le mot-clé ***div***,
- l'opérateur modulo noté par le mot-clé ***mod*** (qui veut dire reste de la division entière),
- opérateurs de relation: <, <=, >, >=, =, ≠ ;

- opérateurs de succession: *succ* et *pred* qui retournent respectivement le successeur et le prédécesseur d'un entier. Par exemple, *succ(-1)= 0* et *pred(2)=1*.
- L'opérateur *ord* qui retourne la même valeur (par exemple : ord(5) = 5),
- fonctions standards telles que racine carrée notée *sqrt*, valeur absolue notée *abs*, etc.

Exemple de déclaration d'une constante de type entier et d'une variable de type entier :

<u>Constante</u>

N = 100

<u>Variable</u>

N1: entier

1.2 Le type réel

Nom : il est désigné par *réel*

Domaine : un sous-ensemble fini des réels.

Représentation des valeurs : elle suit l'écriture de nombres réels avec le changement de la virgule par le point, ou bien, en virgule flottante où deux nombres entiers : mantisse et exposant sont utilisés pour représenter un nombre réel. Exemple : 2.5, -12.55, +2.5, 2E5. Le dernier nombre est écrit en virgule flottante avec une mantisse égale à 2 et un exposant égal à 5, il représente $2*10^5$.

Opérateurs du type réel :

- les opérateurs de relation tels que ceux du type entier,
- les opérateurs arithmétiques: *, +, -, et /
- les fonctions standards: comme *abs*, *sqrt*,..., et les fonctions trigonométriques telles que *arctg*, *sin*, etc...)

Exemple de déclaration d'une constante de type réel et d'une variable de type réel :

<u>Constante</u>

PI = 3.14

<u>Variable</u>

X: réel

Remarques importantes

- La division (entière ou réelle) par zéro n'est pas admise et provoque une erreur de débordement (ou dépassement de capacité).

- Le résultat d'une opération arithmétique peut provoquer une erreur de débordement lorsqu'il sort du domaine de définition du type *entier ou du type réel*.
  Exemple : si a = max_entier-1 et b = max_entier-1, il est évident que le résultat de a* b soit supérieur à max_entier, et provoque un débordement.

1.3 Type booléen ou logique

Nom : il est désigné par *booléen.*

Domaine : il contient les deux valeurs logiques *vrai* et *faux*.

Opérateurs du type logique :

- les opérateurs logiques : <u>et</u>, <u>ou</u> et <u>non</u> .
- Les opérateurs de relations **:** <, >, =, ≠ et nous avons *faux* < *vrai*,
- Les opérateurs *succ* et *pred*, qui retournent respectivement le successeur et le prédécesseur d'un booléen.
  Exemple : *succ(faux)* = *vrai* et *pred(vrai)* = *faux.*
- L'opérateur *ord.* Nous avons *ord(faux)* = 0 et *ord(vrai)* = 1.

Exemple de déclaration d'une constante de type booléen et d'une variable de type booléen :

Constante

V = vrai

Variable

TEST: booléen

1.4 Le type caractère

Nom : il est désigné par *caractère.*

Domaine : les lettres alphabétiques minuscules et majuscules, les caractères numériques, les caractères spéciaux (tels que : ., ?, !, *, <, $, etc.) et le caractère espace (ou blanc). Les caractères sont ordonnés suivant l'ordre des codes machines considérés (ASCII, EBCDIC, etc.). Le plus répandu est le code ASCII (American Standard Code for Information Interchange) dans lequel les caractères représentant les chiffres décimaux et les lettres alphabétiques sont ordonnés et contigus.

Représentation des valeurs : une constante caractère est représentée par un seul caractère mis entre quottes. Le caractère apostrophe est doublé et est placé entre quottes (il est représenté par quatre quottes).

Opérateurs du type caractère :

- La fonction standard *ord* : fournit le code machine correspondant au caractère qui constitue l'argument de *ord.*
- La fonction inverse *chr* : fournit le caractère dont le code est l'argument de *chr.*
- Les opérateurs de relation : <, >, =, ≠, >=, <= car les caractères sont ordonnés.
- les opérateurs *succ* et *pred.*

Exemples:

Ord('A')=65 (en considérant le code machine ASCII), *chr*(65)='A', *succ*('A')='B', *pred*('C')='B'.

Exemple de déclaration d'une constante de type caractère et d'une variable de type caractère :

Constante

ETOILE = '*'

Variable

CAR: caractère

1.5 Le type chaîne

Nom : il est désigné par *chaîne.*

Domaine : l'ensemble des chaînes de caractères que nous pouvons former en juxtaposant des caractères.

Représentation d'une chaîne : une constante de type chaîne est représentée par une juxtaposition de caractères mis entre quottes. Il faut doubler l'apostrophe si celle-ci fait partie d'une chaîne (exemple : 'aujourd''hui').

Opérateurs du type chaîne :

- opérateurs de comparaison de chaînes : <, > , <=, >=, = , ≠
- la fonction *concat* qui sert à concaténer deux chaînes. Exemple :*concat*('Or','an')='Oran'
- la fonction *length* qui fournit la longueur d'une chaîne de caractères donnée. Exemple : *length*('aujourd''hui') = 11.

Exemple de déclaration d'une constante de type chaîne et d'une variable de type chaîne :

Constante

MESSAGE = 'Ce nombre est erroné '

Variable

NOM: chaîne

2. Les types non standards (ou non prédéfinis)

Nous distinguons, parmi les types non prédéfinis, le type énuméré et le type intervalle.

2.1 Le type énuméré

Le type énuméré est défini par l'utilisateur (il n'est pas connu par le processeur). Dans la définition d'un tel type, l'utilisateur énumère une suite finie et ordonnée de valeurs constantes désignées par des noms. Il constitue un *objet de la partie déclaration de nature* **type**. Pour effectuer des traitements avec les valeurs de ce nouveau type, il faut déclarer des variables de ce type. Ce dernier est déclaré comme suit :

```
Type
        Nom-type = ( val1, val2,..., valn)
```

Où *nom-type* représente le nom du type et *val1, val2,..., valn* représentent les constantes de ce type.

Exemple : le type couleur pourrait être déclaré comme suit :

```
Type
        couleur = ( bleu, vert, jaune, rouge)
variable
        CO : couleur     % déclaration d'une variable de ce type %
```

Opérateurs de ce type

- La fonction *ord* qui fournit l'ordre des constantes dans une énumération, par exemple *ord*(bleu) = 0, *ord*(vert) = 1 et ainsi de suite.
- Les opérateurs de relation : <, > et = sont applicables aux valeurs d'un type énuméré.
- Les opérateurs de succession *pred* et *succ*.

Exemple : *succ*(bleu) = vert, *pred*(jaune)= vert.

Remarques

- Le prédécesseur du premier élément et le successeur du dernier élément d'une énumération ne sont pas définis. L'erreur est détectée à l'exécution. Exemple : *succ*(rouge) et *pred*(bleu) sont indéterminés.
- L'énumération définit des constantes: on ne peut donc ni insérer, dans une énumération, des constantes déjà définies, ni énumérer des valeurs d'un type standard (comme par exemple le type entier).

Remarque importante

Les types entier, caractère, booléen et énuméré sont dits des types ordinaux car nous pouvons énumérer leurs valeurs. Nous remarquons aussi qu'ils admettent les opérateurs de succession (*pred* et *succ*) et la fonction *ord* à l'inverse des types réel et chaîne qui ne sont pas des types ordinaux.

2.2 Le type intervalle

Ce type définit un *intervalle* d'un type **ordinal** , par l'indication des bornes inférieure et supérieure de l'intervalle. La borne inférieure doit être inférieure à la borne supérieure.

Le type des constantes qui sont les bornes de l'intervalle précise quel est le type ordinal dont est dérivé l'intervalle. Ce type est déclaré comme suit :

```
Type
        nom-type = borne-inf .. borne-sup
```

Où *nom-type* représente le nom donné à cet intervalle et *borne-inf*, *borne-sup* représentent, respectivement, la borne inférieure et la borne supérieure de l'intervalle.

Exemple :

<u>Type</u>

couleur1 = bleu .. jaune %intervalle dérivé du type énuméré *couleur*%

Jour = 1.. 31 %intervalle dérivé du type entier%

<u>Variable</u>

CO : couleur1 % déclaration d'une variable de type *couleur1* %

Une variable de type intervalle possède toutes les propriétés du type de base dont l'intervalle est dérivé, sauf en ce qui concerne sa valeur, qui doit être comprise entre (ou égale) à ses bornes. Par exemple, la variable CO ne peut prendre que l'une des valeurs bleu, vert, jaune.

Remarque

Le type intervalle peut être considéré comme un type ordinal puisqu'il dérive d'un type ordinal.

3. Compatibilité des types

On dit que deux types sont compatibles :

- s'ils sont identiques;
- ou si l'un est intervalle de l'autre, par exemple : le type entier et le type 0..200;
- ou s'ils sont tous deux intervalles d'un même type, par exemple : le type 0..max_entier et le type 10..350 qui sont tous les deux des intervalles du type entier.

4. Exercices

1- Déclarer les variables suivantes :

- marque d'un ordinateur
- la capacité d'une disquette

- composants d'un ordinateur
- unités d'entre/sortie
- unités d'entrée

2- Détecter les erreurs, si elles existent, dans les déclarations suivantes :

<u>type</u>
(1) ent_nat = 0..max_entier

<u>constante</u>
(2) PI = 3.14
(3) M = 6

<u>variable</u>
(4) 1ENT : ent_nat
(5) N : ent_nat
(6) PI : entier
(7) voyelle1 : ('A', 'E', 'I', 'U', 'O', 'Y')
(8) voyelle2 : (A, E, I, U, O, Y)
(9) voyelle3 : (A, U, O)
(10) chiffre : '0' .. '9'
(11) tranche : 10.50 .. 50.75
(12) interv : inf .. sup
(13) moyenne : réel
(14) log : booléen
(15) ch : chaîne
(16) sexe : (M, F)
(17) val min : ent_nat

CHAPITRE III

LES EXPRESSIONS ET LES INSTRUCTIONS ELEMENTAIRES

Nous rappelons, dans ce chapitre, ce qu'est une expression et présentons les instructions de base qui permettent le transfert d'informations entre objets, et la communication entre un algorithme et son utilisateur. Ce sont *l'affectation* dont le rôle est d'attribuer à un objet une valeur résultant de l'évaluation d'une expression, *la lecture* qui permet l'introduction de données à l'intérieur de la machine, et *l'écriture* qui permet l'affichage des résultats.

1. Les expressions

Il y a plusieurs sortes d'expression : des expressions arithmétiques, des expressions logiques, des expressions de type caractère, des expressions de type chaîne, des expressions de type énuméré et des expressions de type ensemble. Ces dernières ne sont pas abordées ici.

Une expression est formée d'opérandes et d'opérateurs. Un opérande peut être: une constante, une variable, un appel de fonction.

L'évaluation d'une expression faisant intervenir plus d'un opérateur repose sur des règles de priorité entre opérateurs selon l'ordre (du plus prioritaire au moins prioritaire) suivant:

1) opérateurs unaires appliqués à un seul opérande: <u>*non*</u> - +
2) opérateurs multiplicatifs: * / <u>*div*</u> <u>*mod*</u> <u>*et*</u>
3) opérateurs additifs: + – <u>*ou*</u>
4) opérateurs de relation: < <= = ≠ >= >

Remarques

- Pour des opérateurs de même priorité, l'expression est évaluée de gauche à droite.

- L'expression entre parenthèses est évaluée indépendamment des opérateurs placés à droite et à gauche des parenthèses.
- Pour déterminer le type d'une expression il faut vérifier la syntaxe de cette expression (c'est-à-dire la façon de l'écrire), la compatibilité des types des opérandes qui la composent, ainsi que la validité de ses opérateurs. C'est le type des opérandes qui définit le type de l'expression.
- Des opérandes réels et entiers peuvent figurer dans une même expression. Le type de cette dernière est réel.

Exemple

Déterminer le type de l'expression : a-c+b*d/2 sachant que a et b sont des variables de type réel et c et d des variables de type entier.

- la syntaxe de cette expression est correcte ;
- la compatibilité des types est vérifiée : les opérandes réels et entiers peuvent être membres d'une même expression ;
- les opérateurs -, +, * et / sont valides sur les types entier et réel.

Par conséquent, nous pouvons déterminer le type de cette expression. Cette dernière est évaluée comme suit :

b*d	(de type réel)
(b*d)/2	(de type réel)
a-c	(de type réel)
(a-c) + ((b*d)/2)	(de type réel)

2. Les instructions élémentaires

Nous distinguons trois genres d'instructions élémentaires: **l'affectation, la lecture** et **l'écriture**. Nous présentons, dans ce qui suit, chacune de ces actions en utilisant le LA.

2.1 L'affectation

C'est l'action par laquelle nous pouvons attribuer, à une **variable** **v**, une valeur résultant de l'évaluation d'une expression **e**. **Le type de** l'expression doit être compatible avec le type de la variable **v**. **Il y a**

exception si **v** est de type réel et **e** de type entier. Cette instruction est donnée par la syntaxe suivante :

v ← e où ← représente le symbole d'affectation

L'instruction d'affectation signifie : évaluer **e** et ranger le résultat de l'évaluation dans la place mémoire appelée **v**.

Exemple

C ← A+B qui veut dire : évaluer l'expression A+B et mettre le résultat dans la place mémoire appelée C.

2.2 La lecture

C'est l'instruction par laquelle nous pouvons introduire des données à l'intérieur de l'ordinateur. Elle est représentée comme suit par le LA :

<u>lire</u>(x_1, x_2, ...,x_n) où x_1, x_2, ...,x_n sont des variables

L'action **<u>lire</u>**(x_1, x_2, ...,x_n) signifie: mettre dans les places mémoire nommées x_1, x_2, ...,x_n, n données présentes sur l'unité d'entrée de la machine. Au moment de l'exécution, il faudra que l'utilisateur tape au clavier les valeurs des x_i et indique que la saisie de chacune des valeurs est terminée en appuyant sur la touche ' ↵ '.

Exemple

Pour introduire les nombres A et B de l'exemple du paragraphe 2.2 du chapitre 1, nous utilisons l'instruction **<u>Lire</u>**(A,B) qui veut dire mettre dans la place mémoire A, la première donnée tapée au clavier, et mettre dans la place mémoire B, la deuxième donnée tapée au clavier.

2.3 L'écriture

Cette instruction permet de communiquer un résultat ou un message à l'utilisateur (par l'intermédiaire de l'unité de sortie). Elle est représentée comme suit par le LA:

Ecrire($e_1, e_2, \ldots, e_n$) où $e_1, e_2, \ldots, e_n$ *sont des expressions.*

Cette instruction signifie : l'évaluation des expressions $e_1, e_2, \ldots, e_n$ puis leur affichage.

Exemples

écrire('Entrez deux nombres entiers') % message destiné à l'utilisateur pour qu'il tape deux valeurs entières au clavier%

écrire('La somme est égale à ', C) %affichage de la chaîne de caractères: La *somme est* égale à, puis affichage du contenu de la variable C%

écrire(A+B) %évaluation de l'expression A+B puis l'affichage du résultat %

3. Exercices

1- Soit la partie déclaration :

Variable

N1, N2 : entier
X1, X2 : réel
N3 : 0..max_entier
C : caractère
log1, log2 : booléen
chaîne1, chaîne2 : chaîne
revue : (informatique, medecine, electronique, cinema, biologie)

Evaluer et donner le type des différentes expressions suivantes :

(1) N1+ N2 div 4
(2) X1 mod X2 + N3
(3) (N3 < N1 * N2 div 5) et (log1 ou log2)
(4) N2 + N3 < N1 et non log2
(5) Pred (C + 1)
(6) C = 'A' ou C = 'E' ou C = 'U' ou C = 'I'
(7) Chr (X1)
(8) Ord (revue) < N3 + N1
(9) Length (concat (chaîne1, chaîne2))

2- Donner les expressions suivantes en utilisant le LA :

$\frac{AB}{C-4,3D}$ $\frac{-b+\sqrt{b^2-4ac}}{2a}$ $a<x<b$ $x=y$ ou $x=z$ $|x-y|$

3- Détecter les erreurs dans la séquence d'instructions suivante :

constante
(1) R = 20.5

variable
(2) x, y : entier
(3) c : caractère
(4) lire ('c')
(5) lire (R)
(6) y ← *ord* (c)
(7) x ← R/y

4- Définir la suite d'instructions pour les exercices suivants :

1. Lire un nombre réel et l'afficher avec son carré.
2. Calculer la surface S d'un cercle de rayon donné R.
3. Lire un caractère et afficher le caractère qui le suit.
4. Lire une chaîne de caractères et l'afficher avec sa longueur.
5. Faire la permutation de deux nombres entiers x et y.

CHAPITRE IV

LES INSTRUCTIONS STRUCTURÉES (OU STRUCTURES DE CONTRÔLE)

Ce sont des instructions composées qui permettent de spécifier des traitements complexes, à partir d'instructions élémentaires, et d'exprimer la façon dont s'enchaînent ces instructions. Les instructions structurées sont en nombre de trois: la séquence (l'enchaînement naturel), l'alternative (l'enchaînement conditionnel) et l'itération (l'enchaînement répété).

1. La séquence et la notion de bloc

On dit que des instructions forment une séquence, quand la fin d'une instruction déclenche l'exécution de la suivante et ainsi de suite jusqu'à la dernière instruction, constituant ainsi un bloc d'instructions. Ce dernier apparaît à son tour, comme une seule action. Il peut se limiter à une seule instruction.

Schéma

```
          ↓  %marque de début du bloc%
instruction 1
instruction 2
      ...
instruction N

          ↓  %marque de fin du bloc%
```

En particulier, la partie instruction d'un algorithme constitue une séquence délimitée par les mots-clés <u>début</u> et <u>fin</u> et ayant pour en-tête le mot-clé <u>algorithme</u> suivi ou non du nom de l'algorithme (qui est un nom d'objet), suivi de la partie déclaration de l'algorithme.

Exemple

La suite d'instructions suivante, et qui consiste en l'affichage de la somme de deux nombres entiers positifs, constitue une séquence.

<u>écrire</u> ('Entrez deux nombres entiers ')
<u>lire</u> (A, B)
<u>écrire</u> ('La somme de ces deux nombres est de ', A+B)

2. L'alternative

L'exécution d'un bloc d'instructions peut être conditionnée par la *réalisation d'une condition*. C'est pourquoi on a besoin de structures de contrôle conditionnelles qui sont: l'alternative simple et l'alternative complète.

2.1 L'alternative simple

Schéma

```
si  condition        % condition est une
                       expression   booléenne%
alors
   bloc d'instructions
fsi
```

Si la condition est vraie, le bloc d'instructions est exécuté, sinon il est ignoré.

Exemple Suite d'instructions qui permet le calcul d'une racine carrée réelle.

```
si X > 0
alors
   Y ← sqrt(X)
fsi
```

2.2 L'alternative complète

Schéma

```
si  condition        % condition est une
                       expression booléenne%
alors
   bloc d'instructions 1
sinon
   bloc d'instructions 2
fsi
```

Si la condition est vraie, bloc d'instructions 1 est exécuté; si elle est fausse, bloc d'instructions 2 est exécuté.

Exemple: La suite d'instructions qui fait le calcul de la valeur absolue d'un nombre donné N.

```
si N >= 0
alors
   VAL_ABS ← N
sinon
   VAL_ABS ← - N
fsi
```

2.3 Les alternatives imbriquées

Plusieurs alternatives peuvent être imbriquées, il s'agit d'un choix multiple du type:

```
si  condition 1
alors
    bloc d'instructions 1
sinon
        si  condition 2
        alors
            bloc d'instructions 2
        sinon
            si  condition 3
            alors
                bloc d'instructions 3
            sinon
                bloc d'instructions 4
            fsi
        fsi
fsi
```

Exemple

La suite d'instructions qui fait la comparaison de deux nombres entiers positifs N1 et N2, constitue un exemple des alternatives imbriquées.

```
si N1 = N2
alors
   écrire (N1, '=', N2)
sinon
   si N1 < N2
   alors
       écrire (N1, ' < ', N2)
   sinon
       écrire (N2, ' < ', N1)
   fsi
fsi
```

3. L'itération et la notion de boucle

Elle permet de répéter plusieurs fois le même traitement. La répétition du bloc d'actions définit ce que nous appelons des boucles de traitement contrôlées par l'une des trois structures suivantes.

3.1 Boucle répéter

Schéma

```
répéter
    bloc d'instructions
jusqu'à  condition      % condition est une
                          expression booléenne %
```

Nous répétons l'exécution du bloc, jusqu'à ce que la condition soit vraie. Le bloc est exécuté au moins une fois, quelle que soit la condition.

Exemple

La suite d'instructions permettant de vérifier que la valeur d'un nombre tapée au clavier est bien positive.

```
répéter
    écrire ('Entrez un nombre positif')
    lire (nombre)
jusqu'à (nombre > 0)
```

3.2 Boucle Tant que

Schéma

```
tantque  condition      % condition est une
                          expression booléenne%
    faire
        bloc d'instructions
    ffaire
```

Tant que la condition est vraie, on répète l'exécution du bloc d'instructions. Dés qu'elle devient fausse, on sort de la *boucle* et on continue en séquence. Si la condition est fausse initialement, le bloc ne sera jamais exécuté.

Exemple

Calcul de la somme des N premiers nombres entiers positifs en utilisant la formule: S = N + (N - 1) + (N - 2) + ... + 2 + 1.

```
lire (N)
SOM ← 0
tantque N > 0
    faire
        SOM ← SOM + N
        N← N - 1
    ffaire
écrire (SOM)
```

3.3 La boucle Pour

Lorsque l'itération fait apparaître un compteur et que la condition d'arrêt dépend de ce compteur, au lieu des deux schémas précédents, nous pouvons utiliser la boucle Pour.

Schéma

Pour *compteur* **de** *valeur-initiale* **à** *valeur-finale* [**inc** / **dec**] *n*

faire

bloc d'instructions

ffaire

Le bloc d'instructions est exécuté un nombre connu de fois en passant au successeur ou au prédécesseur de la variable *compteur* (qui est appelée aussi variable de contrôle).

compteur est une variable simple de type ordinal. Les valeurs *valeur-initiale* et *valeur-finale* sont des expressions de type compatible avec celui de *compteur*. Elles indiquent, respectivement, la première valeur affectée à *compteur* lors de l'exécution de la première itération, et la dernière valeur qu'aura *compteur* lors de la dernière itération.

Si *valeur-initiale* et *valeur-finale* sont identiques, la boucle est effectuée une fois. Si *valeur-initiale* est supérieure à *valeur-finale*, dans le cas d'un schéma Pour avec **inc**, ou bien si *valeur-initiale* est inférieure à *valeur-finale*, dans le cas d'un schéma Pour avec **dec**, alors la boucle ne s'effectue pas.

n est un entier positif non nul. Il indique le pas d'augmentation (appelé aussi le pas d'incrémentation) ou pas de diminution (appelé aussi le pas de décrémentation) du compteur. Rappelons que l'absence de **inc** *n* ou **dec** *n* sous-entend une incrémentation du compteur d'un pas de 1.

Remarque

L'opération d'augmentation (incrémentation) revient au passage au successeur de la valeur du compteur, et l'opération de diminution (décrémentation) revient au passage au prédécesseur du compteur.

Le schéma Pour décharge le concepteur des opérations suivantes:

- Initialisation de *compteur* à *valeur-initiale*.
- Contrôle de la valeur de *compteur* par rapport à *valeur-finale*.
- Incrémentation ou décrémentation (suivant le cas) de *compteur* à la fin de chaque itération.

Exemple

Donner la suite d'instructions qui calcule la somme suivante :
$S = 1 + 1/3 + 1/5 + 1/7 + 1/9 + \ldots + 1/(2n+1)$ avec $n \geq 0$

```
lire (N)
SOM ← 1
Pour I de 1 à N          %Schéma Pour avec incrémentation du
                           compteur I d'un pas de 1%
    faire
        SOM ← SOM + 1/(2 * I + 1)
    ffaire
écrire ('Cette somme est de ', SOM)
```

ou bien par une solution faisant intervenir un schéma Pour avec un pas différent de 1 :

```
lire (N)
SOM ← 0
P ← 2*N+1
Pour I de 1 à P inc 2          % Schéma Pour avec incrémentation
   faire                         du compteur I d'un pas de 2 %
      SOM ← SOM + 1/I
   ffaire
écrire ('Cette somme est de ', SOM)
```

4. Exercices

Proposer un algorithme pour chacun des exercices suivants :

1. Soient deux nombres X et Y. Ranger dans X la plus petite valeur et dans Y la plus grande.

2. Vérifier si un nombre entier naturel est pair.

3. Déterminer le signe du produit de deux nombres X et Y sans calculer le produit.

4. Soient deux nombres entiers x et y classés par ordre croissant et N un nombre entier quelconque. Afficher ces trois nombres dans l'ordre croissant.

5. Soient trois chaînes c1, c2 et c3. Déterminer et afficher la chaîne c4 obtenue par concaténation de la plus courte chaîne et la plus longue chaîne parmi c1, c2 et c3.

6. Résoudre l'équation du second degré : $Ax^2+Bx+C=0$.

7. Calculer et afficher les sommes suivantes :
   S1 = 1+ 2+3+ ...+100
   S2 = 1+1/2+1/4+1/7+1/11+ ...+1/46
   S3 = 1+1/2-1/3+1/4-1/5+ ...+1/10

8. On considère deux suites :
   $S(a_i) = a_1, a_2, a_3, \ldots, a_n$ et $S(b_i) = b_1, b_2, b_3, \ldots, b_n$.
   Calculer le produit composé :
   $R(c_i) = S(a_i)*S(b_i) = a_1b_1+ a_2b_2+a_3b_3+ \ldots +a_nb_n$.

9. Faire la division de l'entier positif ou nul A par l'entier positif B, à l'aide d'une machine qui ne connaît que: l'affectation, la soustraction, l'addition, la lecture, l'écriture et la boucle tant que.

10. Deux joueurs lancent un dé. Le joueur qui a le plus grand résultat marque un point. On arrête le jeu lorsque l'un des joueurs atteint le score de 11. Simuler ce jeu.

11. Calculer x^n où x représente un nombre réel et n un nombre entier.

12. Calculer e^x en appliquant la formule :
    $e^x = 1+x/1!+x^2/2!+ \ldots +x^n/n!$ avec $x^{n+1}/(n+1)! < \epsilon$.

13. Lire une phrase (caractère par caractère) se terminant par un point et afficher tous les caractères chiffres.

14. Soit une phrase (lue caractère par caractère) se terminant par un point et, où les mots sont séparés par un espace. Compter le nombre de mots contenus dans cette phrase.

CHAPITRE V

LES TYPES DE DONNEES STRUCTUREES STATIQUES OU STRUCTURES DE DONNEES STATIQUES

Un type structuré est composé d'un nombre fixe de données élémentaires reliées d'une certaine façon. Si tous les éléments d'une structure sont de même type, la structure est dite homogène (exemple: tableau et ensemble). Dans le cas contraire, la structure est dite hétérogène (exemple: enregistrement).

1. LES TABLEAUX

Nous étudions ici, les tableaux à une dimension (appelés aussi vecteurs) et les tableaux à deux dimensions (appelés matrices).

1.1 Les vecteurs

Un **tableau à une dimension**, appelé aussi **vecteur**, est une structure de données homogène formée de cellules contigues et d'accès direct.

L'accès direct signifie que nous pouvons obtenir le contenu d'une cellule sans qu'il soit nécessaire de connaître le contenu des cellules précédentes du tableau. Ceci est réalisé à l'aide d'une valeur appelée **indice**. Exemple, l'alphabet français peut être représenté par un vecteur. Les lettres alphabétiques sont les contenus des cellules (ou composantes) de ce vecteur et le rang de la lettre dans l'alphabet représente l'indice de ce vecteur.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Vecteur représentant l'alphabet français

1.1.1 Déclaration d'un vecteur

Pour déclarer un vecteur, il faut utiliser le mot-clé **tableau** et préciser:

- son nom,
- son intervalle d'indiçage qui est de type ordinal, et
- le type des valeurs de ses composantes (ou éléments).

Ainsi un tableau TAB d'indice de type t1 et d'éléments de type t2 est déclaré par :

Type
    TAB= tableau[t1] de t2

Une variable de ce type est **une variable structurée**. Elle regroupe sous un même nom, par exemple T, plusieurs variables élémentaires. Elle est déclarée comme suit:

Variable
    T : tableau[t1] de t2

Une variable V de type TAB est déclarée comme suit :

Variable
    V : TAB

Les deux tableaux T et V, déclarés de deux façons différentes sont de même type : tableau [t1] de t2.

A la déclaration d'une variable tableau, le tableau est créé, mais les composantes n'ont encore aucune valeur ; on dit qu'ils ne sont pas initialisés.

1.1.2 Exemples de tableaux à une dimension

Type
naturel = 0..max_entier
let_maj = 'A' .. 'Z'

situation = (célibataire, marié, divorcé, veuf)
alphabet=tableau[1..26] de let_maj % le type de l'indice est un type ordinal : intervalle%
vect = tableau[situation]de naturel % le type de l'indice est un type ordinal : énuméré%

Le type *vect* a 4 composantes de type entier naturel et le type *alphabet* représente la déclaration du tableau de l'alphabet français.

1.1.3 Accès à un composant du vecteur

Un composant de tableau peut être sélectionné pour un accès direct. Si nous disposons du vecteur T déclaré ci-dessus, et si I est une expression d'indice compatible avec le type t1, alors T[I] donne accès au *I^ème^ composant du tableau* T. T[I] est donc du type t2 et toutes les opérations du type t2 sont possibles avec T[I] qui représente une variable simple.

Exemple

Soit le vecteur alphabet déclaré ci-dessus et I une variable entière définie sur l'intervalle 1.. 26.

alphabet[I] représente la I^ème^ composante du vecteur alphabet. Par exemple, lorsque I = 6, alphabet[6] représente le 6ème élément du tableau alphabet (qui correspond à la lettre F).

1.1.4 Opérations sur un vecteur

Nous pouvons effectuer, sur les *variables de type tableau à une* dimension, certaines opérations comme l'affectation, la comparaison de deux tableaux, la lecture et l'écriture d'un tableau, la recherche séquentielle d'information dans un vecteur, la recherche par dichotomie et la fusion de deux vecteurs. Ces deux dernières opérations ne peuvent s'effectuer que sur des vecteurs triés.

- L'affectation

La seule opération globale que nous pouvons effectuer sur deux tableaux de même type, est l'affectation.

Exemple :
Pour les tableaux T et V (du paragraphe 1.1.1) de même type, nous pouvons effectuer l'affectation globale suivante: T ← V. Elle veut dire affecter les éléments de V aux éléments de T selon leur position respective.

- La comparaison de deux tableaux

Lorsque deux tableaux sont de même type, nous pouvons les comparer. La comparaison se fait élément par élément. Il s'agit de tester l'égalité (ou l'inégalité) entre la première composante du premier vecteur et la première composante du second vecteur puis entre la deuxième composante du premier vecteur et la deuxième composante du second vecteur et ainsi de suite jusqu'à épuisement des deux vecteurs.

- Lecture et écriture d'un vecteur

La lecture (ou écriture) d'une variable de type tableau ne peut se faire qu'élément par élément en parcourant tout le tableau.

Exemple de lecture
L'algorithme suivant effectue la lecture d'un vecteur V de 50 éléments.

```
Algorithme lect_vect
Type
    Vect = tableau [1..50] de réel
Variable
      V : Vect
      I : 1..50
 Début
        Pour I de 1 à 50
           faire
               lire (V[I])
           ffaire
 Fin
```

- Recherche séquentielle (ou linéaire)

Etant donné V un tableau à n éléments non triés et une valeur K du même type que les éléments du tableau V, il s'agit de savoir si cette valeur apparaît dans V et, si elle est présente, à quelle position.

Nous examinons les éléments de V, un à un, en avançant dans le tableau jusqu'à ce que nous rencontrions K ou jusqu'à la fin du vecteur.

Lorsque le tableau V est ordonné, par ordre croissant par exemple, (c'est-à-dire que V[1]≤V[2]≤ ... ≤V[n]) alors, nous recherchons l'élément K en avançant dans V jusqu'à ce que nous rencontrions un élément supérieur ou égal à K, ou jusqu'à la fin du vecteur.

L'inconvénient de cette méthode est que nous risquons de parcourir un tableau, avec un nombre très important d'éléments, sans trouver l'élément cherché. Pour y remédier, il existe une méthode plus efficace qu'on appelle recherche dichotomique.

- Recherche dichotomique

Elle n'est applicable que sur des tableaux triés. Le principe de la recherche dichotomique est de diviser l'intervalle de recherche par 2 à chaque itération. Pour cela, nous procédons de la façon suivante.

Soient D et F les indices des éléments du tableau V trié qui sont les extrémités de l'intervalle dans lequel nous recherchons la valeur K. Nous calculons M, indice de l'élément du milieu.

M ← (D+F) div 2

V [ ] [..........] [ ] [..........] [ ]

D M F

Il y a trois cas possibles:

- K = V[M] l'élément de valeur K est trouvé, la recherche est terminée.
- K < V[M] l'élément de valeur K, s'il existe, se trouve dans l'intervalle [D..M-1]. Il faut prendre M-1 pour nouvelle valeur de F.

➢ K > V[M] l'élément de valeur K, s'il existe, se trouve dans l'intervalle [M+1.. F]. Il faut prendre M+1 pour nouvelle valeur de D.

Nous réitérons ce processus jusqu'à ce que nous ayons trouvé K ou que l'intervalle de recherche soit vide.

- Fusion de deux vecteurs

Soient A et B deux tableaux triés. La fusion de A et B est l'obtention d'un troisième tableau C trié formé des éléments de A et B.

Supposons que A se compose des éléments triés a1≤a2≤...≤ai≤... ≤an et que B se compose des éléments triés b1≤b2≤ ...≤bj≤ ... ≤bm. Nous considérons le couple (ai, bj), 1≤i≤n, 1≤j≤m.

Nous prenons C_k = min(ai, bj), 1≤k≤n+m. L'élément choisi est remplacé par son suivant dans A ou dans B.

Exemple :

A n=6

1	3	5	7	9	10

B m=4

2	4	12	16

C n+m = 10

1	2	3	4	5	7	9	10	12	16

1.2 Matrices ou tableaux à deux dimensions

En mathématiques, un vecteur est un tableau à une dimension, et une matrice est un tableau à deux dimensions. Elle représente une généralisation de la notion d'un vecteur (un vecteur de vecteurs). Par exemple, la feuille de présence mensuelle concernant un groupe de travaux dirigés (TD) d'un certain module, peut être représentée par une matrice, qui à chaque étudiant, fait correspondre les absences et présences durant un mois. En supposant :

- que chaque groupe de TD ne dépasse pas 30 étudiants, et que les étudiants soient numérotés de 1 à 30,

- qu'il s'agit du module d'informatique (4 séances par mois numérotées de 1 à 4), et
- que l'on note par A l'absence d'un étudiant et par P sa présence.

La matrice correspondante est schématisée comme suit :

	1	2	3	4
1	A	A	P	P
2	P	P	P	P
3	A	P	P	P
	.	.	.	
30	P	P	A	A

Matrice (feuille de présence pour un certain mois)

1.2.1 Déclaration d'une matrice

Une matrice possède deux indices : un indice de lignes, et un indice de colonnes. Nous pouvons la définir, aussi, comme un vecteur de vecteurs.

Pour déclarer une matrice, il faut utiliser le mot-clé <u>tableau</u> et préciser:

- son nom,
- le type de ses indices (indice de lignes et indice de colonnes), et
- le type des valeurs de ses composantes (ou éléments).

Nous pouvons décrire une matrice de nom MAT par :

<u>variable</u>

MAT : <u>tableau</u> [t3, t4] <u>de</u> t5 (déclaration 1)

ou bien par : (déclaration 2)

<u>variable</u>

MAT : <u>tableau</u> [t3] <u>de</u> <u>tableau</u>[t4] <u>de</u> t5 % Matrice comme vecteur de vecteur%

Où t3, t4, et t5 indiquent respectivement le type de l'indice de lignes, le type de l'indice de colonnes, et le type des composantes de cette matrice.

1.2.2 Exemples de déclarations de matrices

<u>Type</u>
 Let_maj = 'A' .. 'Z'
 M1 = <u>tableau</u>[1..10] <u>de</u> <u>tableau</u>[let_maj] <u>de</u> entier (1)
 M2 = <u>tableau</u>[1..5] <u>de</u> <u>tableau</u>[-2..2] <u>de</u> réel
<u>Variable</u>
 MAT1: M1
 MAT2 : M2

Ou bien en utilisant la déclaration 1:

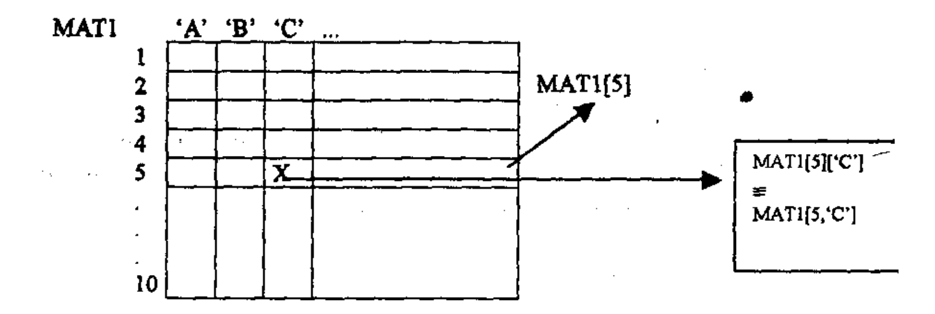
<u>Type</u>
 Let_maj = 'A' ...'Z'
 M1 = <u>tableau</u>[1..10, let_maj] <u>de</u> entier (2)
 M2 = <u>tableau</u>[1..5, -2..2] <u>de</u> réel
<u>Variable</u>
 MAT1: M1
 MAT2 : M2

1.2.3 Accès aux éléments d'une matrice

En utilisant la déclaration 2, si I et J sont deux expressions de type compatible avec t3 et t4 respectivement, alors on a :
MAT [I] qui désigne un tableau à une dimension et,
MAT [I] [J] qui désigne un élément de type t5.
En utilisant la déclaration 1, l'accès à un élément de type t5 se fait par : MAT[*I*, *J*].

Exemple

Accéder à l'élément MAT1[5]['C'] revient à accéder à l'élément MAT1[5, 'C'] et, MAT1 [5] désigne une variable de type <u>tableau</u> [let_maj] <u>de</u> entier, voir schéma ci-dessous.



1.2.4 Opérations sur une matrice

La seule opération globale possible est l'affectation. D'autres traitements sont également possibles tels que la recherche d'une valeur dans une matrice, la lecture ou l'écriture d'une matrice etc. Ces traitements nécessitent le parcours de la matrice élément par élément.

L'algorithme suivant montre la lecture d'une matrice MAT de 50 lignes et 100 colonnes.

```
Algorithme lec_mat
Type
    M = tableau [1..50, 1..100] de entier
Variable
    MAT : M
        I : 1..50
        J :1.. 100
 Début
       Pour I de 1 à 50
          faire
              Pour J de 1 à 100
                faire
                     Lire (MAT[I, J])
                ffaire
          ffaire
 Fin
```

2. ENREGISTREMENT

2.1 Définition

Un article (appelé aussi enregistrement ou record en anglais) est une structure composée d'un nombre fixe d'éléments qui peuvent être de types différents. Les éléments d'un article sont appelés les **champs** de l'article, et peuvent être à leur tour des structures (tableaux, articles, ...).

2.2 Déclaration de l'article

Un type article de nom T se *déclare comme* :

```
Type
   T = article
          C1 : t1
          C2 : t2
           ...
          Cn : tn
        fin
```

Les composants C1, C2, ..., Cn sont appelés champs de l'article et sont délimités par les mots-clés **article** et **fin**. Les ti sont les types correspondant aux Ci et peuvent être *structurés* ou non.

Un objet *art* de type T, se déclare comme :

```
Variable
      art : T
```

2.3 Exemples de déclarations d'enregistrements

1) La déclaration d'un nombre complexe peut se faire comme suit :

```
type
   complexe = article              %Exemple d'article avec champs
                                     de type simple%
                 p_real : réel
                 p_im : réel
              fin
```

```
variable
     nbr_comp : complexe
```

2) La déclaration d'une commande de produits peut se faire comme suit :

```
type
  date = article
          jour : 1..31
          mois : 1..12
          année : 1900..2100
        fin
  designation = article
                  nom : chaîne
                  prenom : chaîne
                fin
  commande = article                     % commande est un article
                                            composé d'articles%
                   D : date              % Le champ D est de type
                                            article%
                   client : designation  %designation est un champ
                                            de type article%
                   num-com , num-prod : 0..max-entier
                   montant : réel
                fin
Variable
     CDE : commande         % Déclaration d'une variable de type
                              commande %
```

3) La déclaration du PV de délibérations de juin, pour les étudiants du TC SETI, peut se faire comme suit :

```
type
  ETUD = article
            nom, prenom : chaîne
            notes : tableau[1..9] de réel       % Le champ notes est
                                                  de type tableau %
            moy-generale : réel
            resultat : (admis, ajourné)
         fin
variable
     PV : tableau[1..600] de ETUD       % Exemple de tableau dont
                                          les éléments sont des
                                          articles%
```

2.4 Désignation des champs de l'article

Pour désigner un champ d'article, nous indiquons le nom de la variable article suivi d'un point et suivi du nom du champ. Par exemple, pour désigner la partie réelle du nombre complexe *nbr_comp* de l'exemple 1, nous écrivons : **nbr_comp.p_real**.

Pour désigner le mois de la commande CDE de l'exemple 2, on écrit: **CDE.D.mois**.

Pour désigner la 3ème note du 2ème étudiant sur le PV de délibération de l'exemple3, nous écrivons : **PV[2].notes[3]**.

Pour désigner le 2ème champ du 1er étudiant sur le PV de délibération de l'exemple3, nous écrivons : **PV[1].prenom**.

2.5 Opérations sur le type enregistrement

Il existe des opérations sur les variables de type enregistrement prises globalement, et des opérations sur les champs de l'enregistrement.

a. Opérations sur les enregistrements

La seule opération possible sur les enregistrements de même type est l'affectation.

Exemple :
Si *CDE1* et *CDE2* sont deux variables de type *commande*, nous pourrons effectuer l'affectation : CDE1 ← CDE2.

Remarque1

Nous ne pouvons pas lire ou écrire globalement, un enregistrement. Pour le faire, il faudra lire ou écrire chaque champ qui le compose.

Exemple
Pour lire l'enregistrement ***complexe*** de l'exemple 1, on écrit :
lire (nbr_comp.p_real, nbr_comp.p_im) et non pas **lire** (nbr_comp).

Remarque2

Nous ne pouvons pas tester l'égalité (=) ou la différence (≠) de deux enregistrements de même type d'une façon globale. La comparaison doit se faire champ par champ : le premier champ du premier enregistrement est comparé au *premier champ du deuxième* enregistrement, le deuxième champ du premier enregistrement est comparé au deuxième champ du deuxième enregistrement, et ainsi de suite jusqu'à la fin des deux enregistrements.

b. Opérations sur les champs

Les opérations possibles sur les champs sont celles réalisables sur les variables du type de ces champs.

Exemple
CDE.montant est un champ de type réel. On peut donc effectuer l'opération :
CDE.montant ← CDE.montant * 4 / 100, etc.

3. LES ENSEMBLES

Nous nous limitons dans cette partie du cours au type ensemble de PASCAL. Contrairement aux articles et aux tableaux qui peuvent avoir plusieurs niveaux d'imbrications les ensembles ne permettent pas de réaliser des combinaisons de structures : il existe des ensembles de valeurs de type ordinal, et nous ne pouvons pas construire d'ensembles de tableaux, d'articles ou d'ensembles.

3.1 Définition

Le type ensemble permet de représenter l'objet mathématique ensemble dont la théorie a été largement développée. **Un ensemble est une collection non ordonnée d'éléments sur lesquels on peut effectuer les opérations mathématiques classiques telles que l'appartenance, l'union, l'intersection, la différence, etc.**

3.2 Déclaration du type ensemble

Un type ensemble particulier de nom *nom_ens* est défini à partir du type de base *t* de ses éléments (qui est un type ordinal) et à l'aide du constructeur **ENSEMBLE** comme il est montré dans la partie déclaration ci-après. La définition du type de base t donne les bornes et la capacité de l'ensemble.

```
Type
   nom_ens = ensemble de t
```

Exemples

```
Type
   color = (blanc, rouge, noir)
   couleur = ensemble de color
   chiffres = ensemble de 0..9
Variable
   C : couleur
```

3.3 Expression d'ensemble

Nous ne pouvons pas définir de constantes de type ensemble, mais nous pouvons définir des expressions d'ensembles et des ensembles constants.

L'ensemble constant : [] représente l'ensemble vide. Si a, b, c sont des constantes de même type, [a, b, c] représente l'ensemble constant comprenant les trois valeurs a, b, c. Si *a et b sont deux constantes* de même type, [a..b] représente l'ensemble constant comprenant les valeurs : a, succ(a), ..., pred(b), b. Si b< a l'ensemble est vide.

La variable C, déclarée dans les exemples du paragraphe 3.2, peut prendre les valeurs : [], [blanc], [rouge], [noir], [blanc, rouge], [blanc, noir], [rouge, noir] et [blanc, rouge, noir] (les crochets sont utilisés pour représenter un ensemble constant).

Exemple

Les ensembles [3, 8, 6] et [1..4, 9] représentent chacun un ensemble de type chiffres.

3.4 Opérations sur les ensembles

Les opérations possibles sur des constantes ou des variables du même type ensemble sont :

- L'union notée +

A + B = L'ensemble des éléments appartenant à A **ou** appartenant à B.

Exemple : [3, 4, 8] + [2, 4] = [2, 3, 4, 8]

- L'intersection notée *

A * B = l'ensemble des éléments appartenant à A **et** appartenant à B.

Exemple : [3, 4, 8] * [2, 4] = [4]

- La différence notée –

A – B = l'ensemble des éléments appartenant à A **et n'appartenant** pas à B.

Exemple : [3, 4, 8] - [2, 4] = [3, 8]

- La comparaison

Il existe deux opérations de comparaison : **l'égalité notée = et l'inégalité notée ≠.**

Nous disons que A = B si tous les éléments de A sont dans B et tous les éléments de B sont dans A indépendamment de leur position.
Exemple : [3, 8] = [8, 3] donne la valeur vrai.

Nous disons que A ≠ B s'il existe un et un seul élément de A qui n'est pas dans B.
Exemple : [3, 4] ≠ [3], [4] ≠[3, 5] donnent toutes les deux la valeur vrai.

- Inclusion et contenance

Nous disons que A est inclus dans B (noté **A ≤ B**) si tous les éléments de A appartiennent à B. Nous disons que A contient B (noté **A ≥ B**) si tous les éléments de B appartiennent à A.
Exemple : [3] ≤ [3, 4] et [0..9] ≥ [4] donnent toutes les deux la valeur vrai.

- L'appartenance notée <u>dans</u>

Pour tester si un élément de même type que les éléments d'un ensemble donné, appartient à cet ensemble, nous utilisons l'opérateur **<u>dans</u>**.
Exemple : 3 **<u>dans</u>** [3, 6] donne la valeur vrai, et 3 **<u>dans</u>** [4, 9] donne la valeur faux.

- L'affectation

Comme toute variable, une variable de type ensemble doit être initialisée avant toute utilisation à l'aide de l'opérateur d'affectation.
Exemples : C ← [], C ← [blanc, rouge]

3.5 Exemple d'utilisation des ensembles

Lire une suite de chiffres non nuls, se terminant par un zéro. Déterminer et afficher l'ensemble des chiffres qui apparaissent dans cette suite. Par exemple, si nous introduisons la suite 3511390, il faut former l'ensemble {1,3,5,9} et afficher les quatre valeurs de cet ensemble.

```
Algorithme Construction

Type
   T= ensemble de 1..9
Variable
   Ens : T
   chiffre : 0..9

Début
% Lecture de la suite de chiffres et construction de l'ensemble %
  Ens ← []
  écrire ('Introduire une suite de chiffres se terminant par zéro')
  lire (chiffre)
  tantque chiffre ≠ 0
   faire
     Ens ← Ens + [chiffre]
     lire (chiffre)
   ffaire
% Affichage des éléments de l'ensemble %
  écrire ('L''ensemble des chiffres qui apparaissent dans cette suite
          est : {')
  Pour chiffre de 1 à 9
    faire
      si chiffre dans Ens
      alors
        écrire (chiffre, ' ')
      fsi
    ffaire
  écrire ('}')
Fin
```

Remarque

L'algorithme construction nous montre l'opération de construction d'un ensemble ainsi que l'affichage de ses éléments.

4. Exercices

Proposer un algorithme pour chacun des exercices suivants :

1- On dispose d'un vecteur de 50 notes. Calculer et afficher la moyenne de ces notes.

2- Déterminer la valeur maximale, d'un vecteur à n valeurs entières, avec son rang.

3- Soit V un vecteur à 100 valeurs réelles. Extraire, à partir de ce vecteur, deux autres vecteurs P et N. Le premier contiendra les éléments positifs de V et le second, les éléments négatifs de V. Afficher les éléments des vecteurs P et V.

4- Soient 2 vecteurs à 50 valeurs entières chacun. Comparer ces deux vecteurs.

5- On dispose d'un vecteur de n entiers naturels. Ranger les nombres pairs au début du vecteur et les nombres impairs à la fin.

6- Soit un vecteur de 100 noms rangés par ordre alphabétique. On voudrait savoir si un nom donné se trouve dans ce vecteur (utiliser le principe de dichotomie).

7- Calculer le produit des éléments d'une matrice à nxm éléments réels.

8- Mettre à zéro les deux diagonales d'une matrice carrée de 50 lignes et 50 colonnes.

9- Soit MAT une matrice carrée d'ordre 100. Trouver le nombre des éléments, au dessus de la diagonale principale, tels que : $10 < Mat[i, j] < 30$, et calculer la somme de ces éléments.

10- Soit une matrice M de 200 lignes et 100 colonnes, à valeurs entières. Déterminer la ligne dont la somme des éléments est maximale.

11- La feuille de soins médicaux regroupe les renseignements suivants concernant l'assuré salarié : nom, prénom, date et lieu de naissance, adresse personnelle, nom et adresse de l'employeur , et mode de paiement. Décrire ce type.

12- Un nombre complexe est défini par : a+ib.

a) Lire deux complexes et afficher leur produit.
b) Comparer deux nombres complexes.

13- Dans une bibliothèque, on dispose de 1000 fiches. Les informations que l'on pourrait lire sur la fiche d'un ouvrage de la bibliothèque sont les suivantes :
code, auteur (nom et prénom), titre, éditeur, et année.
Afficher les auteurs qui ont édité en l'an 2001 chez édition Chihab.

14- Lire une phrase (caractère par caractère) se terminant par un point. Compter le nombre d'apparitions des caractères de ponctuation.

15- Lire une phrase (caractère par caractère) se terminant par un point. Déterminer et afficher l'ensemble des lettres alphabétiques minuscules qui n'apparaissent pas dans cette phrase.

16- Lire une phrase (caractère par caractère) se terminant par un point. Déterminer et afficher l'ensemble des caractères chiffres qui apparaissent dans la phrase.

CHAPITRE VI

PROCEDURES ET FONCTIONS

Dans les algorithmes que nous avons construits jusqu'à présent, les données sont obtenues par l'intermédiaire de l'instruction ***lire*** : elles sont transmises à l'algorithme par l'utilisateur qui les saisit par clavier. Les résultats, quant à eux, sont communiqués à l'utilisateur par l'exécution de l'*instruction écrire*. Avec ces deux catégories d'instructions, le processeur effectue l'échange d'information entre l'utilisateur et l'algorithme. Nous allons, dans ce chapitre, étudier un autre mode d'échange d'informations qui s'effectue entre un algorithme et un autre algorithme. L'utilisation d'un tel mécanisme facilite la résolution de problèmes complexes.

1. Notion de tâche (ou module)

Dans le cas d'un problème simple, il est aisé d'écrire une suite d'instructions constituant **un seul algorithme** pour le résoudre totalement. Par contre si le problème est plus complexe, son algorithme serait très long, difficile à développer, à comprendre et à modifier. C'est la raison pour laquelle nous sommes amenés à décomposer le problème en **tâches indépendantes** (ou modules), qui à leur tour peuvent être décomposées jusqu'à obtenir des tâches simples à résoudre et pouvant être développées, parallèlement, par différentes personnes. Nous appelons ce principe de décomposition **analyse descendante**.

Exemple

Considérons le problème qui consiste à faire les délibérations en juin pour les étudiants du tronc commun SETI. Ce problème, étant complexe, il est difficile de le résoudre par un seul algorithme. Lui trouver une solution revient à le décomposer en tâches et à associer à chacune d'elles un algorithme. Ces tâches peuvent être : saisie des notes par module, calcul de la moyenne générale, affichage de la liste des admis, et enfin affichage de la liste de rattrapage par module.

2. Nature d'une tâche

Une tâche peut être : une **procédure** ou une **fonction.** Une procédure est une instruction non élémentaire (regroupant une suite d'actions élémentaires) créée par le concepteur. Une fonction est une tâche qui réalise un calcul non élémentaire et produit une valeur unique.

Exemple de procédure : l'instruction *échanger deux nombres entiers A et B* est une instruction non élémentaire pour le processeur et doit être explicitée par le concepteur en écrivant une procédure.

Exemple de fonction : la tâche *faire la moyenne de deux nombres* se fait par un calcul et produit une valeur unique (une moyenne qui est une valeur réelle). Elle doit être spécifiée par une fonction.

3. Description d'une tâche

Une tâche est composée d'un en-tête, d'une partie déclaration locale et d'une partie instruction (représentant la suite d'instructions nécessaires à sa réalisation). **L'en-tête** commence par l'un des mots-clés: **procédure**, ou **fonction** (suivant le cas), suivi du nom de la tâche (qui est un nom d'objet), puis par une liste de paramètres formels. Un paramètre formel est une variable qui est utilisé aussi dans la partie instruction de la tâche et, à travers laquelle, deux algorithmes peuvent communiquer entre eux. Il est défini par son **rôle** : donnée (s'il s'agit d'un paramètre d'entrée), résultat (s'il s'agit d'un paramètre de sortie) ou donnée-résultat (s'il s'agit d'un paramètre servant d'entrée et de sortie à la fois), son **nom** et son **type**. Une fonction n'admet pas de paramètre résultat ni de paramètre donnée-résultat. C'est le nom de la fonction qui joue le rôle du paramètre résultat. Par conséquent, le type du résultat de la fonction doit figurer dans son en-tête.

Dans notre LA : E définit un paramètre donnée, S définit un paramètre résultat, et ES définit un paramètre donnée-résultat (E, S, et ES sont soulignés : ce sont des mot-clés). Une tâche peut nécessiter des objets autres que les paramètres, dans sa partie instruction. Ces objets constituent **la partie déclaration locale** de la tâche.

4. Représentation de l'en-tête d'une tâche

Nous présentons, dans ce qui suit, l'en-tête d'une procédure ainsi que celui d'une fonction avec des exemples.

a. En-tête de procédure : il est donné par la syntaxe suivante.

Procédure nom-procédure (paramètres-formels)
Où nom-procédure = nom de la procédure
Paramètres-formels = paramètres de la procédure

Par exemple, l'en-tête de la tâche *échanger deux nombres entiers* est donné comme suit, en utilisant notre LA :

Procédure ECHANGE (ES A,B: entier) % A et B sont tous les deux des paramètres donnée-résultat, ils sont précédés par ES%

b. En-tête de fonction : il est donné par la syntaxe suivante.

fonction nom-fonction (paramètres-formels) : type-fonction
Où nom-fonction = nom de la fonction
Paramètres-formels = paramètres de la fonction
type-fonction = type du résultat retourné par la fonction

Par exemple, l'en-tête de la tâche *moyenne de deux nombres* est donné comme suit, en utilisant notre LA :

Fonction MOYENNE (E X,Y: réel) : réel % X et Y sont tous les deux des paramètres donnée, ils sont précédés par E %

Les algorithmes suivants représentent les descriptions entières des tâches : *échanger deux nombres entiers* et *moyenne de deux nombres.*

Procédure ECHANGE (ES A, B : entier)   % A et B sont des paramètres formels %

%Partie déclaration locale de la procédure%

variable
    C : entier

%partie instruction de la procédure%

Début
  C ← B
  B ← A   %Les paramètres formels A et B figurent dans la partie instruction de la procédure%
  A ← C
Fin

Fonction MOYENNE (E X, Y : réel) : réel

%Partie déclaration locale vide%

%partie instruction de la fonction%

Début
  MOYENNE ← (X + Y) / 2   %Affectation du résultat de la fonction au nom de la fonction%
Fin

Remarque

La partie déclaration locale d'une tâche peut être vide, c'est le cas de la fonction MOYENNE. Celle de la procédure ECHANGE comprend une seule variable : C.

5. Déclaration d'une tâche

Une tâche est vue comme un objet par l'algorithme qui l'utilise. Elle doit donc être déclarée dans la partie déclaration de ce dernier.

a. Cas d'une procédure

La déclaration d'une procédure définit sa nature et son nom. La procédure ECHANGE est déclarée comme suit :

<u>Procédure</u>

ECHANGE

b. Cas d'une fonction

La déclaration d'une fonction définit sa nature, son nom et son type. La fonction MOYENNE est déclarée comme suit :

<u>Fonction</u>

MOYENNE : réel

6. Appel d'une tâche

L'appel d'une tâche se fait dans l'algorithme qui utilise cette tâche. Ce dernier constitue l'algorithme **appelant** et la tâche constitue l'algorithme **appelé**. La syntaxe de l'appel est la suivante :

Nom-de-tâche (paramètres-effectifs)

Avec nom-de-tâche = nom de la tâche appelée
paramètres-effectifs = les paramètres de l'algorithme appelant, ils sont séparés par une virgule.

Un paramètre effectif est une variable utilisée dans l'algorithme appelant et qui sert dans la communication d'information entre deux algorithmes (communication entre l'appelant et l'appelé). Lors de l'appel, les paramètres effectifs (de l'appelant) viennent se substituer aux paramètres formels (qui se trouvent dans l'en-tête de l'appelé). Les paramètres **effectifs** doivent correspondre aux paramètres **formels** en <u>nombre</u>, <u>type</u> et <u>position</u>.

Remarque importante

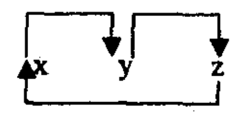
L'appel de la tâche vu ci-dessus, constitue une **instruction** dans le cas d'une procédure, et une **expression** dans le cas d'une fonction.

7. Exemples illustratifs

Nous montrons, à travers les deux exemples suivants, comment se fait l'appel d'une procédure et celui d'une fonction.

a. Cas d'une procédure :

Il s'agit de définir l'algorithme qui fait la permutation circulaire de trois nombres entiers. L'une des solutions, à ce problème, est d'utiliser la procédure ECHANGE. L'appelant dans cet exemple est l'algorithme qui fait la permutation circulaire et, l'appelé est représenté par la procédure ECHANGE. La permutation de x, y et z, est schématisée comme suit :



__Algorithme__ permutation_circulaire %algorithme appelant %

__Variable__
X, Y, Z : entier
__Procédure__ %Déclaration de la procédure ECHANGE dans la
ECHANGE partie déclaration de l'algorithme appelant %

__Début__
__écrire__ ('Entrez trois nombres entiers pour faire leur permutation circulaire')
__lire__(X, Y, Z)
ECHANGE(Y, Z) %l'instruction d'appel de la procédure ECHANGE
avec les paramètres effectifs Y et Z %
ECHANGE(X,Y) % l'instruction d'appel de la procédure ECHANGE
avec les paramètres effectifs X et Y %
__écrire__ ('Après permutation circulaire, les trois nombres sont : ', X,', ', Y,
' et ', Z)
__Fin__

Procédure Echange (**ES** A, B : entier)   %algorithme appelé%

Variable
    C : entier

Début
C ← B
B ← A
A ← C
Fin

Au moment de l'appel de la procédure ECHANGE, la première fois, la valeur de Y (paramètre effectif) est transférée dans A (paramètre formel). De même la valeur de Z est transférée dans B et la procédure ECHANGE est exécutée (l'échange est effectué entre Y et Z). Lors du deuxième appel de la procédure ECHANGE, la valeur de X (paramètre effectif) est transférée dans le paramètre formel A. De même la valeur de Y est transférée dans B et la procédure ECHANGE est exécutée une seconde fois (l'échange est effectué entre X et Y). Pour les deux appels, la procédure ECHANGE, après exécution, fournit le résultat de l'échange dans les mêmes paramètres A et B (A et B sont des paramètres donnée-résultat).

Remarque

Nous constatons, sur cet exemple, l'un des avantages de la décomposition d'un problème en tâches. C'est celui de définir une tâche (la procédure ECHANGE) lorsque le même type de traitement doit être répété plusieurs fois (deux fois dans l'exemple), à différents endroits de l'algorithme appelant. En définissant ce traitement sous forme d'une tâche, nous ne le décrirons qu'une fois (la procédure ECHANGE est décrite une seule fois) et nous effectuerons un appel pour chaque demande d'exécution.

b. Cas d'une fonction

Il s'agit de définir l'algorithme « moy_etud » qui fait le calcul de la moyenne des deux partiels, pour tous les étudiants au niveau d'un même module, et l'affichage du nom de l'étudiant suivi de sa

moyenne. L'une des solutions à ce problème est d'utiliser la fonction MOYENNE. L'appelant dans cet exemple est l'algorithme **moy_etud**. Il fait appel à la fonction **MOYENNE** et à la procédure **saisie_note**. Cette dernière permet de vérifier que 0≤note≤20.

```
Algorithme moy_etud

Variable
      NOM_ETU : chaîne
      P1, P2 : réel           %représentent, respectivement, note du premier partiel
                                 et note du deuxième partiel %
      I, NBR_ETU : 1.. 600
Fonction                                     %Déclaration de la fonction MOYENNE
                                                dans la partie déclaration de
      MOYENNE :réel                             l'algorithme appelant : moy_etud %

Procédure
      Saisie_note                             %Déclaration de la procédure
                                                 saisie_note dans la partie déclaration
                                               de l'algorithme appelant : moy_etud %
Début
  écrire('Entrez le nombre total des étudiants du module')
  lire(NBR_ETU)
  pour I de 1 à NBR_ETU
   faire
     écrire('Entrez le nom de l''étudiant')
     lire(NOM_ETU)
     saisie_note(P1)              % Appel de la procédure saisie_note pour saisir
                                     P1 et vérifier que 0≤P1≤20 %
     saisie_note(P2)              % Appel de la procédure saisie_note pour saisir
                                     P2 et vérifier que 0≤P2≤20 %
     écrire(NOM_ETU, ' ', MOYENNE (P1, P2))   %L'appel de la fonction
                                                 moyenne dans une expres-
                                                 sion en utilisant les
                                                paramètres effectifs P1 et P2 %

   ffaire
fin

Fonction MOYENNE (E X, Y : réel) : réel

Début
  MOYENNE ← (X + Y )/ 2
Fin
```

```
Procédure saisie_note (S note :réel)

Début
    répéter
        écrire ('Saisir la note')
        lire (note)
    jusqu'à (0<= note) et (20>=note)
fin
```

Lorsque l'expression *MOYENNE (P1, P2)* est rencontrée (au niveau de l'algorithme *moy_etu*) les valeurs des paramètres effectifs P1 et P2 sont transférées, respectivement, dans les paramètres formels X et Y et la fonction MOYENNE est exécutée. Le résultat de la fonction correspond au résultat de l'évaluation de l'expression MOYENNE(P1, P2).

Remarque :

- Le LA, utilisé ici, n'impose pas de contrainte d'emplacement de l'algorithme appelé par rapport à l'algorithme appelant.
- Une tâche (procédure ou fonction) peut faire appel à d'autres tâches si le traitement le nécessite.

8. Exercices

1- Définir la fonction MIN3 qui détermine le minimum de trois nombres en utilisant une fonction MIN2 qui fournit le minimum de deux nombres.

2- Calculer C^p_n = n !/p !(n-p) !, en utilisant une fonction qui détermine le factoriel d'un nombre entier naturel N.

3- Déterminer le plus petit multiple commun (PPCM), de deux nombres entiers naturels A et B, en appliquant la formule :
$$PPCM(A, B) = \frac{A * B}{PGCD(A, B)}$$
PGCD(A, 0) = A
PGCD(A, B) = PGCD(B, A mod B)

4- Un nombre d'Armstrong est un entier naturel qui est égal à la somme des cubes de ses chiffres. Exemple : $153 = 1^3 + 5^3 + 3^3$. Concevoir un algorithme qui permet d'afficher tous les nombres d'Armstrong inférieurs à un entier naturel donné. Utiliser une fonction qui fournit le cube d'un chiffre et une fonction qui vérifie si un nombre donné est un nombre d'Armstrong.

5- Concevoir un algorithme qui utilise la procédure ECHANGE (définie au paragraphe VI.4) pour classer trois nombres entiers par ordre croissant.

6- Soit une matrice de 100 lignes et 200 colonnes à valeurs réelles.

a- Définir le type MAT de cette matrice.

b- Concevoir une procédure qui détermine la position de la valeur minimale de cette matrice.

7- Une matrice fournit l'état des absences de 30 étudiants (numérotés de 1 à 30) pendant 20 séances de TD (numérotées de 1 à 20). Ecrire un algorithme qui :

- réalise la saisie des données de cette matrice, en utilisant une procédure. Cette procédure devra vérifier que la valeur saisie, de l'élément, est un des 2 caractères : A (absent) ou P (présent),
- affiche, le nombre d'absences de chaque étudiant, en utilisant une fonction qui détermine le nombre d'absences d'un étudiant de numéro donné.

8- Le vecteur de type RESULTAT répertorie les résultats d'un examen auquel ont participé 200 candidats. Chaque composant de ce vecteur comprend le nom du candidat et sa note. Ecrire un algorithme qui :

- fait la saisie des données du vecteur (en utilisant une procédure),
- calcule la moyenne de l'examen (en utilisant une fonction),
- affiche la liste des candidats ayant une note supérieure ou égale à la moyenne de l'examen.

9- Soient deux nombres rationnels R1 et R2. On désire calculer la somme de R1 et R2 et afficher le résultat sous sa forme réduite (utiliser une procédure qui saisit un nombre rationnel).

10- Soit une matrice carrée d'ordre 100 dont les éléments sont des chiffres. Ecrire une procédure qui détermine et affiche l'ensemble des chiffres situés en dessous de la diagonale principale.

11- A partir de deux phrases (lues caractère par caractère) se terminant par un point, on désire construire deux vecteurs. Les éléments de chaque vecteur représentent le nombre d'apparition de chaque lettre de l'alphabet dans la phrase. Ecrire un algorithme qui :

- construit ces deux vecteurs (en utilisant une procédure),
- détermine l'ensemble des lettres alphabétiques apparaissant dans la première phrase, et l'ensemble des lettres alphabétiques apparaissant dans la deuxième phrase (en utilisant une procédure),
- détermine et affiche l'ensemble des lettres alphabétiques communes aux deux phrases,
- détermine et affiche l'ensemble des lettres alphabétiques figurant dans la première phrase et ne figurant pas dans la deuxième phrase.

Exemple

Phrase1 : il a préparé son examen.

Phrase 2 : il a réussi.

Vecteur 1 (correspondant à phrase1)

3	0	0	0	4	...	1	0	0	1	1	2	1	2	0	2	1	0	0	...	1	0	0
a	b	c	d	e		i	j	k	l	m	n	o	p	Q	r	s	t	u		x	y	z

vecteur 2 (correspondant à phrase 2)

1	0	0	0	1	...	2	0	0	1	0	0	0	0	0	1	2	0	1	...	0	0	0
a	b	c	d	e		i	j	k	l	m	n	o	p	Q	r	s	t	u		x	y	z

ens 1 = {a, e, i, l, m, n, o, p, r, s, x}
ens 2 = {a, e, i, l, r, s, u}
ens 3 = {a, e, i, l, r, s}
ens 4 = {m, n, o, p, x}

Corrigé des exercices

Corrigé des exercices du chapitre I

1.

Nous savons qu'un octet = 8 bits. Comme la taille du mot de cette machine est de 16 bits = 2 octets, nous pouvons déduire la taille de cette mémoire en octets. Elle est égale à la capacité de la mémoire en mots * 2 = 1048576 * 2 = 2097152octets.
1 kilo-octet =1024 octets, nous pouvons donc déduire la capacité de cette mémoire en kilo-octets. Elle est égale à la capacité de la mémoire en octets / 1024 = 2097152 / 1024 = 2048 kilo-octets.

2.

Les deux droites D1 et D2 sont définies par les équations :
Y = A1X + B1, Y = A2X + B2.

a. Les objets constituant la partie déclaration de ce problème sont : A1, B1, A2 et B2 (qui représentent les coefficients des deux droites), X et Y (qui représentent les coordonnées du point d'intersection des deux droites).

b. La suite d'actions qui permet de donner une solution à ce problème est :

- saisir A1, B1, A2 et B2 (donner des valeurs à A1, B1, A2 et B2)
- tester les coefficients :
    - si A1 et A2 sont égaux et, B1 et B2 sont égaux, nous avons une infinité de points d'intersections (les deux droites sont confondues)
    - si A1 et A2 sont égaux et, B1 et B2 sont différents, nous n'avons pas de point d'intersection (les deux droites sont parallèles)
    - si A1 et A2 sont différents (il existe un point d'intersection) :

- calculer $\frac{B2-B1}{A1-A2}$ et mettre le résultat dans X
- calculer A1 X + B1 et mettre le résultat dans Y
- afficher les coordonnées du point d'intersection X et Y

3.

a. Les objets constituant la partie déclaration de ce problème sont : sac, boules, compteur (permettant de compter le nombre de boules rouges contenues dans le sac).

b. La suite d'actions qui permet de donner une solution à ce problème est :

- mettre zéro dans le compteur
- répéter
    - tirer une boule du sac
    - tester la couleur de la boule : si la boule est rouge alors ajouter 1 au compteur

  jusqu'à ce que le sac soit vide
- afficher le contenu du compteur

4.

a. Les objets constituant la partie déclaration sont : file, client, carte d'identité nationale (CNI), chèque, guichet.

b. La suite d'actions qui permet de donner une solution à ce problème est :

répéter

- prendre la CNI et le chèque du client présent devant le guichet
- si CNI et chèque sont conformes, alors servir le client
- retirer le client de la file
- passer au client suivant

jusqu'à ce que la file soit vide

Corrigé des exercices du chapitre II

1.

variable

marque_ordinat: chaîne
capacit_disq : réel
composant_ordinat : (processeur, memoire, unit_entree, unit_sortie)
unit_es : (clavier, souris, scanner, camera, imprimante, ecran_visu, traceur)
unit_ent : clavier..camera

2.

- Erreur en ligne 4. Un nom doit commencer par une lettre or *1ENT* commence par un chiffre.

- Erreur en ligne 6. *PI* a déjà été déclarée comme une constante réelle; nous ne pouvons pas la redéfinir comme une variable de type entier.

- Erreur en ligne 7. Nous ne pouvons pas énumérer des valeurs d'un type standard. Le type standard considéré, ici, est le type caractère.

- Erreur en ligne 9. Nous ne pouvons pas insérer, dans une énumération, des constantes déjà définies. Les constantes : *A, O, U* de *voyelle 3* ont déjà été définies dans *voyelle 2* (en ligne 8).

- Erreur en ligne 11. Les bornes d'un intervalle doivent être de type ordinal, or les bornes 10.5 et 50.75 sont de type réel (qui n'est pas un type ordinal).

- Erreur en ligne 12. Les bornes *inf* et *sup* de l'intervalle *interv* ne sont pas déclarées.

- Erreur en ligne 16. Nous ne pouvons pas insérer, dans une énumération, des constantes déjà définies. La constante *M* dans *sexe* est déjà déclarée (en ligne 3).

- Erreur en ligne 17. Le nom *val min* est incorrect. Le caractère espace (ou blanc) ne doit pas figurer dans le nom d'un objet.

Corrigé des exercices du chapitre III

1.

- La syntaxe de l'expression en ligne 1 est correcte. N1, N2 et 4 sont des opérandes de même type (entier) ; +, div sont des opérateurs valides sur le type entier. Cette expression est évaluée en appliquant la priorité des opérateurs comme suit :
  N2 div 4 de type entier, puis (N1+(N2 div 4)) de type entier et qui représente le type de l'expression N1+N2 div 4.

- L'expression, en ligne 2, présente une syntaxe correcte, mais elle comprend un opérateur non valide sur les opérandes qui la composent. En effet, X1 et X2 sont de type réel et l'opérateur mod n'est pas valide sur le type réel, par conséquent, nous ne pouvons pas donner le type de cette expression.

- La syntaxe de l'expression en ligne 3 est correcte. Dans la sous-expression (N3<N1*N2 div5) : N1, N2, N3et 5 sont de type compatible avec le type entier. Les opérateurs <, *, div sont valides sur le type entier. Dans la sous-expression (log1 ou log2), log1 et log2 sont de type booléen ; l'opérateur ou est valide sur le type booléen. L'évaluation de l'expression initiale se fait de gauche à droite en commençant par la première sous-expression (N3<(N3<N1*N2 div5) qui est évaluée comme suit :
  N1*N2 de type entier
  (N1*N2 div5) de type entier
  (N3<((N1*N2)div5)) de type booléen, ensuite nous évaluons la sous-expression (log1oulog2) qui est de type booléen, et enfin, l'expression : ((N3<((N1*N2)div5)) et (log1oulog2)) qui est de type booléen.

- La syntaxe de l'expression en ligne 4 est correcte. L'application de la priorité des opérateurs nous amène à l'évaluation de : non log2, puis la sous-expression (N1 et (non log2)). Nous constatons, sur cette dernière, que l'opérateur et est appliqué sur un opérande de type entier (N1) et un autre de

type booléen (non log2). Cette sous-expression est erronée : l'opérateur et doit s'appliquer sur deux opérandes booléens. Par conséquent, le type de l'expression globale ne peut être déterminé.

- L'expression donnée en ligne 5 est erronée : C est de type caractère et l'opérateur + est non valide sur le type caractère.

- L'expression donnée en ligne 6 est erronée. L'application de la priorité des opérateurs, à cette expression, donne : C = ('A' ou C) = ('E' ou C) = ('U' ou C) ='I'. Nous remarquons que l'opérateur ou est appliqué à des opérandes de type caractère, or l'opérateur ou n'est pas valide sur le type caractère.

- L'expression donnée en ligne 7 est erronée. La fonction *chr* n'admet pas un argument de type réel (X1 est de type réel).

- La syntaxe de l'expression en ligne 8 est correcte. *revue* est un opérande de type énuméré. La fonction *ord* est valide sur le type énuméré et fournit un entier naturel. Les opérandes de cette expression sont tous de type compatible avec le type entier. Les opérateurs +, < sont valides sur le type entier. L'évaluation de cette expression se fait comme suit :

  - ord(revue) de type 0..4 (compatible avec le type entier)
  - N1+N2 de type entier
  - ((ord(revue)) < (N1+N2)) de type booléen .

- La syntaxe de l'expression en ligne 9 est correcte. chaîne1 et chaîne2 sont de type chaîne. La fonction *concat* admet deux arguments de type chaîne et fournit un opérande de type chaîne. La fonction *length* est valide sur le type chaîne. L'expression initiale est évaluée comme suit :

- (*concat*(chaîne1, chaîne2)) de type chaîne
- (*length*(*concat*(chaîne1, chaîne2))) de type 0..max_entier.

2.

Nous donnons, dans le tableau suivant, chacune des expressions mathématiques avec l'*expression correspondante utilisant* le langage algorithmique.

L'expression mathématique	L'expression utilisant le langage algorithmique
$\frac{AB}{C-4,3D}$	A*B/(C-4.3*D)
$\frac{-b+\sqrt{b^2-4ac}}{2a}$	(-b+sqrt(b*b-4*a*c))/(2*a)
$a<x<b$	(a<x) et (x<b)
x=y ou x=z	(x=y) ou (x=z)
$\lvert x-y \rvert$	abs(x-y)

3.

- Erreur en ligne 4 : 'c' représente la lettre alphabétique minuscule c qui est une constante, et on ne peut pas lire une constante.

- Erreur en ligne 5 : R est déclaré en ligne 1 comme une constante réelle de valeur 20.5, nous ne pouvons pas changer sa valeur. Par conséquent, l'instruction lire(R) est non autorisée.

- Erreur en ligne7 : Dans une instruction d'affectation, le type de l'expression à droite du symbole d'affectation doit être compatible avec le type de la variable à gauche du symbole. L'expression R/y est de type réel. Nous ne pouvons pas affecter une expression de type réel à une variable de type entier (x est une variable de type entier).

4.

Nous montrons, à travers ces exercices, l'utilisation des instructions élémentaires.

1. écrire ('Saisir un nombre réel')
lire (x)
écrire ('Le carré du nombre ', x, ' est ', x*x)

2. Pour le calcul de la surface d'un cercle, nous appliquons la formule : surface = ΠR^2, avec $\Pi = 3{,}14$ *et* R = *rayon du cercle.*

écrire ('Saisir le rayon d'un cercle, pour calculer la surface de ce cercle')
lire (R)
écrire ('La surface d'un cercle de rayon ', R, ' est ', 3.14*R*R)

3. Pour afficher le caractère qui suit un caractère donné, nous utilisons la fonction succ .

écrire ('Saisir un caractère')
lire (c)
écrire ('Le caractère qui suit le caractère ', c, ' est ', succ(c))

4. La longueur d'une chaîne est donnée par la fonction length.

écrire ('Saisir une chaîne de caractère')
lire (ch)
écrire ('La longueur de la chaîne ', ch, ' est ', length (ch))

5. La permutation de deux nombres x et y, nécessite une variable supplémentaire z pour sauvegarder l'un des deux nombres.

écrire ('Saisir deux nombres entiers')
lire (x, y)
z ← x
x ← y
y ← z
écrire ('Après permutation, les deux nombres sont ', x, ' et ',y)

Corrigé des exercices du chapitre IV

1.

Nous n'effectuons l'échange entre les deux nombres x et y que si la valeur de y est inférieure à la valeur de x. Nous avons besoin d'une variable supplémentaire pour effectuer l'échange comme dans la solution de l'*exercice 5 du chapitre* III.
Nous montrons, à travers cet exercice, l'utilisation de l'alternative simple.

```
Algorithme ranger

 Variable
   x, y, z : réel

 Début
  écrire('Rangement dans x de la plus petite valeur et dans y, la plus
         grande')
  écrire ('Saisir 2 nombres réels')
  lire (x, y)
  si x> y
  alors
    z ← x
    x ← y
    y ← z
  fsi
  écrire ('Après rangement, les deux nombres sont ', x, ' et ', y)
 Fin
```

2.

Un nombre pair est divisible par 2 (le reste de la division entière de ce nombre par 2 vaut 0). C'est l'opération mod qui permet de fournir ce reste.
Nous montrons, à travers cet exercice, l'utilisation de l'alternative complète.

```
Algorithme test_pair

 Variable
    N : 0..max_entier

 Début
  écrire('Tester si un nombre est pair')
  écrire ('Saisir  un nombre entier naturel')
  lire (N)
  si N mod 2 = 0
  alors
    écrire (N, ' est un nombre pair')
  sinon
    écrire (N, ' n''est pas un nombre pair')
  fsi
 Fin
```

3.

Si les deux nombres sont positifs ou négatifs, le résultat sera positif. La négation de cette condition donnera un résultat négatif.
Nous montrons, à travers cet exercice, l'utilisation de l'alternative complète.

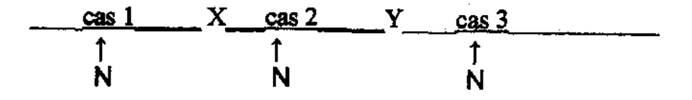
```
Algorithme signe

 Variable
  x, y: réel

 Début
  écrire('Signe du produit de deux nombres sans calculer le produit')
  écrire ('Saisir 2 nombres réels')
  lire (x, y)
  si ((x> 0) et (y>0)) ou ((x<0) et (y<0))
  alors
    écrire ('Le signe du produit de ', x, ' par ', y, ' est positif')
  sinon
    écrire ('Le signe du produit de ', x, ' par ', y, ' est négatif')
  fsi
 Fin
```

4.

X et Y étant classés par ordre croissant. Il s'agit de positionner N par rapport à X et Y de façon à conserver l'ordre croissant entre les 3 nombres X, Y et N. Trois cas se présentent comme le montre le schéma suivant.



Cas 1 : $N < X$
Comme $X < Y$, il résulte que les 3 nombres classés sont N, X et Y.

Cas 2 : $N > X$ et $N < Y$
Comme $X < Y$, il résulte que les 3 nombres classés sont X, N et Y.

Cas 3 : $N > Y$
Comme $X < Y$, il résulte que les 3 nombres classés sont X, Y et N.
La boucle répéter utilisée dans l'algorithme sert à obliger l'utilisateur à donner deux nombres classés par ordre croissant.
Nous montrons à travers cet exercice un exemple d'utilisation des alternatives imbriquées.

```
Algorithme classer

 Variable
  X, Y, N : entier

 Début
 écrire ('Positionner un nombre par rapport à 2 nombres entiers classés par
         ordre croissant')
 répéter        % Obliger l'utilisateur à donner 2 nombres classés par ordre
                    croissant %
  écrire ('Saisir 2 nombres entiers classés par ordre croissant')
  lire (X, Y)
 jusqu'à Y > X
 écrire ('Saisir un troisième nombre entier')
 lire (N)
```

```
% Positionner N par rapport à X et Y qui sont déjà classés par ordre
    croissant %
 si N < X      % cas 1 %
 alors
   écrire ('Les 3 nombres classés sont ', N, ' ', X, ' et ', Y)
 sinon
   si N < Y    % cas 2 %
   alors
     écrire ('Les 3 nombres classés sont ', X, ' ', N, ' et ', Y)
   sinon       % cas 3 %
     écrire ('Les 3 nombres classés sont ', X, ' ', Y, ' et ', N)
   fsi
 fsi
Fin
```

5.

Nous utilisons les variables min et max pour sauvegarder, respectivement, la plus courte chaîne et la plus longue chaîne. Nous commençons par comparer les longueurs des deux premières chaînes (en utilisant la fonction *length*). La plus courte des deux chaînes est sauvegardée dans min, et la plus longue dans max. Ensuite, nous comparons chacune des longueurs de la chaîne min et de la chaîne max avec celle de la troisième chaîne. La plus courte chaîne est mémorisée dans min et la plus longue dans max. Enfin, nous terminons en affichant la chaîne obtenue par concaténation (en utilisant la fonction *concat*) de la plus courte et de la plus longue chaîne.

```
Algorithme concaténation
 Variable
  c1, c2, c3, c4, min, max : chaîne

 Début
  écrire ('Concaténation de la plus courte chaîne et de la plus longue chaîne
          parmi trois chaînes')
  écrire ('Introduire 3 chaînes de caractères')
  lire (c1, c2, c3)
```

```
% Comparaison de la longueur de la première chaîne et celle de la 2ème
   chaîne %
si length (c1) < length (c2)
alors
 min ← c1              % La sauvegarde de la plus courte chaîne dans la
                          variable min %
 max ← c2              % La sauvegarde de la plus longue chaîne dans la
                          variable max %
sinon
 min ← c2
 max ← c1
fsi
si length (min) < length (c3)   % Comparaison de la longueur de la chaîne
                                   min et celle de la 3ème chaîne %
alors
 si length(c3) < length(max)   % Comparaison de la longueur de la chaîne
                                   max et celle de la 3ème chaîne %
 alors
  c4 ← concat (min, max)
 sinon
  c4 ← concat (min, c3)
 fsi
sinon
 c4 ← concat (c3, max)
fsi
écrire ('La concaténation de la plus courte chaîne et de la plus longue
        chaîne donne', c4)
Fin
```

6.

Nous présentons, ici, un autre exemple d'utilisation d'alternatives imbriquées. C'est la résolution d'une équation du second degré avec étude de tous les cas possibles :

Si A = 0 et B = 0 et C = 0 alors nous avons une infinité de solutions.

Si A = 0 et B = 0 et C ≠ 0 alors il n'y a pas de solution.

Si A = 0 et B ≠ 0 alors il existe une solution égale à –C/B.

Si A ≠ 0, alors nous calculons $\Delta = B^2 - 4\,AC$

Si Δ < 0 alors il n'y a pas de solution

Si $\Delta = 0$ alors il existe une racine double égale à –B/(2*A)
Si $\Delta > 0$ alors il existe deux racines distinctes :
$X1 = (-B-\sqrt{\Delta})/2A$ et $X2 = (-B+\sqrt{\Delta})/2A$.

```
Algorithme equation

 Variable
  A, B, C, X1, X2, DELTA : réel

 Début
  écrire ('Résolution d''une équation du second degré')
  écrire ('Saisir les coefficients d''une équation du second degré')
  lire(A, B, C)
  si A = 0
  alors
   si B = 0
   alors
     si C = 0
     alors
      écrire ('Il existe une infinité de solutions')
     sinon
      écrire ('Il n''y a pas de solution')
     fsi
   sinon
    X1 ← -C/B
    écrire ('Il existe une solution : ', X1)
  fsi
 sinon
  DELTA ← B*B-4*A*C
  si DELTA > 0
  alors
   X1 ← (-B - sqrt(DELTA))/(2*A)
   X2 ← (-B + sqrt(DELTA))/(2*A)
   écrire( 'L''équation admet 2 racines: X1= ', X1, ' et X2 = ', X2)
  sinon
   si DELTA = 0
   alors
    X1 ← -B/(2*A)
    écrire( 'L''équation admet une racine double égale à ', X1)
```

```
    sinon
      écrire ('Cette équation n''admet pas de solution')
    fsi
   fsi
  fsi
Fin
```

7.

Cet exercice montre l'utilisation des itérations.

- S1 nous permet d'avoir la somme des 100 premiers nombres entiers *naturels (le zéro étant exclu)*. Nous initialisons la variable S1 à zéro et nous utilisons une boucle Pour. La valeur du compteur I est initialisée à 1 (valeur correspondant au premier nombre entier naturel à sommer). A chaque nouvelle itération, la valeur du compteur I (qui correspond à l'entier naturel suivant) est ajoutée à la somme S1. Le processus s'arrête lorsque la valeur finale 100 du compteur I est ajoutée à S1. La deuxième instruction écrire dans l'algorithme sert à afficher la chaîne de caractère *La somme des 100 premiers entiers naturels est :* , suivie de la valeur de la somme S1.

```
Algorithme somme1

 Variable
  I : 1..100
  S1 : 0..max_entier

 Début
  écrire ('Somme des 100 premiers nombres entiers naturels')
  S1 ← 0                 % Initialisation de la somme S1 à zéro %
  pour I de 1 à 100
   faire
    S1 ← S1 + I          % Ajouter l'entier naturel I à la somme S1 et mettre
                           le résultat dans S1 %
   ffaire
  % Affichage %
  écrire ('La somme des 100 premiers entiers naturels est : ', S1)
 Fin
```

- Nous constatons que le terme de rang i+1 est obtenu en ajoutant i au dénominateur du terme de rang i. Nous commençons à partir du rang 1 Nous notons pat J le dénominateur et nous l'initialisons à 1. I est initialisé à 0. A chaque itération, le terme 1/J est ajouté à la somme S2 (initialisée à 0) à condition que J ne dépasse pas 46.
Nous montrons à travers cet exercice l'utilisation de l'itération tant que.

```
Algorithme somme2

 Variable
  I : 0..max_entier
  J : 1..46
  S2 : 0..max_entier

 Début
  écrire ('Calcul de la somme : 1+1/2+1/4+1/7+1/11+...+1/46')
  I ← 0
  J ← 1
  S2 ← 0
  tantque J <= 46
   faire
    S2 ← S2 + 1/J
    I ← I +1
    J ← J + I
   ffaire
  écrire ('1+1/2+1/4+1/7+1/11+........+1/46 = ', S2)
 Fin
```

- Nous remarquons, à partir du deuxième terme de la somme S3, que les termes qui ont un dénominateur pair sont positifs, et les termes qui ont un dénominateur impair sont négatifs. Pour calculer cette somme, on l'initialise avec le premier terme qui est égal à 1 et on utilise une boucle Pour avec le compteur I variant de 2 (2ème terme) à 10. Les termes 1/I sont ajoutés à la somme si I est pair, ils sont retranchés sinon.
Nous montrons, à travers cet exercice l'utilisation de structures de contrôle imbriquées : l'alternative complète à l'intérieur d'une boucle pour.

```
Algorithme somme3

 Variable
  I : 2..10
  S3 : 1..max_entier

 Début
  écrire ('Calcul de la somme : 1+1/2-1/3+1/4-1/5+...+1/10' )
  S3 ← 1
  pour I de 2 à 10
   faire
    si I mod 2 = 0
    alors
     S3 ← S3 + 1/ I
    sinon
     S3 ← S3 – 1/I
    fsi
  ffaire
  écrire ('1+1/2-1/3+1/4-........+1/10 = ', S3)
 Fin
```

8.

Il suffit de constater dans ce produit composé qu'il faut former le produit ai*bi pour i variant de 1 à n, et l'ajouter à une somme (appelée R dans la solution présentée) initialisée à 0. Le nombre d'itération étant connu, le schéma Pour est conseillé.

```
Algorithme prod_comp

 Variable
  I, n : 1..300
  R, a, b: réel

 Début
  écrire ('Calcul du produit composé de deux suites')
  écrire ('Introduire le nombre de termes des suites')
  lire (n)
  R ← 0
```

```
  pour I de 1 à n
   faire
    écrire ('Saisir le ', i, 'ème élément de la première suite et celui de la
              deuxième suite')
    lire (a, b)
    R ← R+ a*b
   ffaire
  % Affichage %
  écrire('Le produit composé de ces 2 suites est de : ', R)
 Fin
```

9.

D'une manière générale, diviser A par B, c'est trouver Q et R positifs ou nuls, tels que : A=B*Q+R avec R< B (reste< diviseur). Cette opération revient à faire des soustractions successives, par exemple, diviser 13 par 4 revient à faire :

13-4 = 9   première soustraction
 9-4 = 5   deuxième soustraction
 5-4 = 1   troisième soustraction

nous nous arrêtons car 1< 4 (reste< diviseur). Le nombre de *soustractions représente le quotient* et le résultat de la dernière soustraction représente le reste. Nous initialisons R à A et Q à 0.
Nous montrons, à travers cet exercice l'utilisation des deux schémas tant que et répéter.

```
Algorithme division

 Variable
  A, Q, R : O..max_entier
  B : 1..max_entier

 Début
  écrire ('Division entière par soustractions successives')
  écrire ('Introduire un nombre entier naturel')
  lire (A)
  répéter
   écrire ('Saisir un nombre naturel non nul')     % Eviter un  diviseur nul %
   lire (B)
  jusqu'à  B>0
```

```
R ← A          % Initialisation du reste R à A %
Q ← 0          % Initialisation du quotient Q à 0 %
tantque R >= B
 faire
  R ← R –B     % Calcul de la nouvelle valeur du reste R %
  Q ← Q + 1    % Incrémenter Q de 1 après chaque soustraction R-B %
 ffaire
% Affichage %
écrire ('La division de ', A, ' par ', B, ' donne un quotient égal à ',Q, ' et
        un reste égal à ', R)
Fin
```

10.

Il s'agit, dans cet exercice de comparer la valeur du dé lancé par le premier joueur (D1) et celle du dé lancé par le deuxième joueur (D2). Si D1 est supérieure à D2 alors nous ajoutons 1 au score du premier joueur. Si D2 est *supérieure à D1 alors nous ajoutons 1 au score* du deuxième joueur. Nous répétons ce processus jusqu'à ce que l'un des deux joueurs atteigne un score de 11 points.
Cet exercice montre l'utilisation de l'alternative imbriquée dans la boucle répéter.

```
Algorithme jeu

Variable
 D1, D2 : 1..6             % Dés du premier et du deuxième joueur %
 SCOR1, SCOR2 : 0..11      % score du premier et score du deuxième
                              joueur %
Début
 SCOR1 ← 0        % Initialisation du score du joueur 1 à 0 %
 SCOR2 ← 0        % Initialisation du score du joueur 2 à 0 %
 répéter
  écrire ('Introduire la valeur du dé du premier joueur et celle du dé du
          deuxième joueur')
  lire (D1, D2)
  si D1 > D2
  alors
   SCOR1 ← SCOR1 + 1  % Incrémentation du score du premier joueur %
```

```
    sinon
      si D1 < D2
      alors
        SCOR2 ← SCOR2 + 1   % Incrémentation du score du deuxième
                                                  joueur %
      fsi
    fsi
  jusqu à  (SCOR1 = 11)  ou  (SCOR2 = 11)
  % Affichage %
  si SCOR1 = 11
  alors
    écrire ('Le gagnant est le joueur 1')
  sinon
    écrire ('Le gagnant est le joueur 2')
  fsi
Fin
```

11.

Nous étudions ici, les différents cas de x^n avec x réel et n entier (positif, négatif ou égal à nul).

1. Si $x = 0$ et $n > 0$, alors $x^n = 0$.
2. Si $x = 0$ et $n \leq 0$, alors x^n est non définie.
3. Si $x \neq 0$ et $n \geq 0$, alors nous calculons x^n qui est égale à x*x*...*x (n fois). Nous utilisons pour cela une variable P que nous initialisons à 1 et une boucle Pour. A chaque itération, nous effectuons le produit P*x. Au bout de n itérations, nous obtenons la valeur de x^n. Elle est mémorisée dans la variable P.
4. Si $x \neq 0$ et $n < 0$, alors nous nous ramenons au 3ème cas en remplaçant n par –n et x par 1/x.

Cet exercice montre un mélange de structures (alternatives imbriquées et boucle Pour).

```
Algorithme puissance

 Variable
  P, x : réel
  n, i  : entier
```

```
Début
 écrire ('Calcul de x puissance n avec x réel et n entier')
 écrire ('Introduire un nombre réel')
 lire (x)
 écrire ('Introduire l''exposant')
 lire (n)
 si x = 0
 alors
  si n > 0
  alors
   écrire (x, ' puissance ', n, ' est = ', 0)
  sinon
   écrire (x, ' puissance ', n , ' est non définie')
 fsi
 sinon
 P ← 1
 si  n < 0
 alors
  n ← - n
  x ← 1/x
 fsi
 pour i de 1 à n
  faire
   P ← P * x
  ffaire
 écrire (x,' puissance ', n, ' est = ', P)
 fsi
Fin
```

12.

Nous utilisons le principe de calcul d'une somme (par exemple la somme des *100* premiers entiers naturels de la solution de l'exercice 7 de ce chapitre). Nous remarquons que le terme de rang i+1 est obtenu à partir du terme de rang *i* par une multiplication comme suit : $x^{i+1}/(i+1)! = (x^i/i!)*(x/i+1)$. Le terme de rang i+1 est ajouté à la somme EXPO initialisée à 0, s'il est supérieur en valeur absolue à ε. Nous négligeons les termes inférieurs en valeur absolue à ε.
Nous montrons à travers cet exercice un autre exemple de l'utilisation de la boucle tant que.

Algorithme exponentiel

Constante
EPSILON = 0.001

Variable
X, Y, EXPO : réel
I : 0..*max_entier*

Début
écrire ('Calcul de expo(x) = 1+x/1 !+x*x/2 !+...')
écrire ('Introduire un nombre réel pour calculer son exponentielle')
lire (X)
I ← *0*
Y ← 1
EXPO ← 0
tantque abs(Y) > EPSILON
faire
EXPO ← EXPO + Y
I ← I + 1
Y ← Y * X / I
ffaire
écrire ('Exponentiel de ', X , ' vaut ', EXPO)
Fin

13.

Nous utilisons dans cet exercice, une boucle répéter pour lire tous les caractères de la phrase un à un jusqu'à la rencontre du caractère point indiquant la fin de la phrase. Tout caractère lu est testé. Il est affiché si c'est un caractère chiffre c'est-à-dire compris entre '0' et '9'.

Algorithme affich_chiffre

Variable
C : caractère

Début
écrire ('Affichage des caractères chiffres apparaissant dans une phrase')
écrire ('Introduire une phrase caractère par caractère et se terminant par un point')

```
  répéter
  lire (C)
   si (C >= '0') et (C<= '9')   % Tester si le caractère lu est un caractère
                                              chiffre %
   alors
    écrire (C, ' ')
   fsi
  jusqu'à C = '.'          % Le point indique la fin de la phrase %
Fin
```

14.

Nous utilisons un compteur de mots (*compt*) que nous initialisons à 0. Nous introduisons la phrase caractère par caractère. Un mot est formé lorsque nous rencontrons un espace ou bien un point, par conséquent le compteur de mot est incrémenté de 1. Le point indique la fin de la phrase et aussi la formation du dernier mot de la phrase. Nous utilisons deux boucles répéter imbriquées. La boucle interne sert à compter les mots, et la boucle externe sert à lire tous les caractères de la phrase jusqu'au point final.

```
Algorithme compte_mots

 Variable
  C : caractère
  compt : 0...max_entier

 Début
  écrire ('Compter le nombre de mots dans une phrase')
  compt ← 0
  écrire ('Saisir une phrase caractère par caractère et se terminant par un
         point')
  répéter          % Boucle externe %
   répéter          % Boucle interne %
    lire (C)
   jusqu'à (C = '.') ou (C = ' ')
   compt ← compt + 1      % Incrémentation du compteur %
  jusqu'à C = '.'
  écrire ('Le nombre de mots dans cette phrase est ', compt)
 Fin
```

Corrigé des exercices du chapitre V

1.

Nous calculons la moyenne de 50 notes par la formule : somme des 50 notes/50. Comme ces notes constituent les éléments d'un vecteur, nous utilisons une boucle Pour, pour la saisie et la somme des 50 éléments du vecteur. Le compteur I sert à parcourir tous les éléments du vecteur V, et *som* sert à récupérer la somme de tous les éléments de V. La moyenne des notes est représentée par le contenu de la variable *som* divisé par 50 (*som*/50) dans la 2ème instruction d'écriture de l'algorithme.

```
Algorithme moy_vect

 Variable
    V : tableau[1..50] de réel
    I : 1..50
    som : réel

 Début
  % Saisie des notes (dans un vecteur) et calcul de leur somme %
  écrire('Saisir  les 50 notes')
  som ← 0
  pour I de 1 à 50
   faire
     lire ( V[I] )
     som ← som + V[I]
   ffaire
 % Affichage de la moyenne des notes %
 écrire ('La moyenne des 50 notes est égale à :  ', som/50)
 Fin
```

2.

Comme il s'agit d'un vecteur de n éléments, il faut donc fixer une taille maximale pour le vecteur et laisser le choix à l'utilisateur de donner une taille au vecteur ne dépassant pas la taille maximale fixée.

La méthode adoptée est de supposer que la valeur maximale du vecteur est représentée, initialement, par le premier élément du vecteur et que son rang (représenté par *Rg* dans l'algorithme) est 1. Il s'agit ensuite de parcourir le vecteur à partir du 2ème élément (en utilisant une boucle Pour) et de comparer la valeur maximale à chacun des éléments du vecteur. A chaque itération, si la valeur maximale (représentée par la variable *Max* dans l'algorithme) est inférieure à la valeur de l'élément *V[I]* du vecteur, alors Max prend la valeur de ce dernier et *Rg* prend, pour valeur, l'indice *I* de l'élément *V[I] du vecteur.*

```
Algorithme Max_Vect

Variable
 V : tableau[1..100] de entier   % Taille maximale du vecteur est fixée à
                                           100%
 Rg, I, n: 1.. 100
 Max : entier

Début
   % Saisie %
   écrire ( 'Donner la taille du vecteur')
   lire (n)                              % n ne doit pas dépasser 100 %
   écrire (' Saisir les éléments du vecteur')
   pour I de 1 à n
    faire
       lire (V[I])
    ffaire
% Calcul de l'élément maximal avec son rang %
Max ← V[1]
Rg ← 1
pour I de 2 à n
  faire
      si V[I] > Max
      alors
          Max ← V[I]
          Rg ← I
      fsi
 ffaire
```

```
% Affichage %
écrire ('La valeur maximale du vecteur est : ',Max,' et son rang est : ',
        Rg)
Fin
```

3.

Le vecteur initial (représenté par *V* dans l'algorithme) est parcouru en utilisant une boucle Pour. A chaque itération, la valeur de l'élément du vecteur *V* est testée. Si elle est positive ou nulle, nous la stockons dans le vecteur des éléments positifs appelé *P* et nous avançons dans le vecteur *P* pour recevoir le prochain élément positif ou nul du vecteur V. Dans le cas contraire, nous la stockons dans le vecteur des éléments négatifs appelé *N* et nous avançons dans le vecteur *N* pour recevoir le prochain élément négatif du vecteur *V*.

NB : Nous supposons que les éléments du vecteur *V* sont saisis.

```
Algorithme Extraction

Type
 vect = tableau [1..100] de réel

Variable
 V, P, N: vect
 I, J, K : 1..100

Début
  %Extraction%
  J ← 1                  % J représente l'indice du vecteur P %
  K ← 1                  % K représente l'indice du vecteur N %
  pour I de 1 à 100      % I représente l'indice du vecteur V %
   faire
      si V[I] >= 0
      alors
         P[J] ← V[I]  % stockage de l'élément positif ou nul de V dans P %
         J ← J+1        % avancer dans le vecteur P %
```

```
      sinon
         N[K] ← V[I]   % stockage de l'élément négatif de V dans N %
         K ← K+1       % avancer  dans le vecteur N %
      fsi
   ffaire
% Affichage %
 écrire ('Les éléments du vecteur P sont : ')
 pour I de 1 à J-1
   faire
      écrire ('P(', I, ')= ', P[I])
   ffaire
 écrire ('Les éléments du vecteur N sont : ')
 pour I de 1 à K-1
   faire
      écrire ('N(', I, ')=' , N[I])
   ffaire
 Fin
```

4.

Nous tenons à noter ici, qu'il est impossible d'utiliser une boucle Pour, étant donné que le nombre d'itérations n'est pas connu à l'avance.

La méthode consiste à comparer les deux vecteurs élément par élément (une comparaison globale est impossible). Pour cela, nous utilisons une variable booléenne (*val_bool*) qui a pour rôle d'arrêter le processus de comparaison dés que nous rencontrons, dans le premier vecteur, un élément différent de l'élément de même indice dans le deuxième vecteur. Dans le cas d'égalité des deux éléments comparés, *val_bool* prend la valeur *vrai*, elle prend la valeur *faux* sinon. Nous répétons le processus de comparaison jusqu'à épuisement des deux vecteurs (exprimé par la condition I>50) ou bien jusqu'à rencontrer deux éléments différents après comparaison (exprimé par la condition *val_bool= faux)*. L'égalité des deux vecteurs est déduite de la condition *I>50*, leur différence est déduite de la condition *val_bool= faux.*

NB : Nous supposons que les éléments des deux vecteurs A et B sont saisis.

Algorithme Comparer_2vect

```
Variable
  A, B : tableau [1..50] de entier
  I : 1..50
  val_bool : booléen

Début
 % Comparaison%
 I ← 1
 répéter
    si A[I] = B[I]          % Comparaison d'un élément du 1er vecteur avec
    alors                     l'élément du même indice dans le 2ème vecteur %
       val_bool ← vrai
    sinon
       val_bool ← faux
    fsi
    I ← I+1
 jusqu'à (I > 50) ou (val_bool = faux)
 % Affichage %
 si I > 50
 alors
   écrire ('Les deux vecteurs sont égaux')
 sinon
   écrire ('Les deux vecteurs ne sont pas égaux')
 fsi
Fin
```

5.

Nous parcourons le vecteur *vect* élément par élément. A chaque itération, nous testons si l'élément est impair (c'est à dire le reste de la division entière par 2 ≠ 0). Dans l'affirmative, nous permutons l'élément du vecteur *vect* d'indice *i* (initialisé à 1) et l'élément du vecteur vect d'indice *j* (initialisé à N) et nous décrémentons *j* par pas de 1. Dans le cas contraire (l'élément est pair), nous passons à l'élément suivant du vecteur en incrémentant l'indice *i*. Nous répétons ce processus jusqu'à ce que les indices *i* et *j* coincident.
NB : Les éléments et la taille du vecteur *vect* sont supposés saisis.

Algorithme Rangement

```
Variable
  vect : tableau[1..100] de 0..max_entier
  N, i : 1..100
  j : 0..100
  X : 0..max_entier

Début
  j ← N
  i ← 1
  répéter
    si vect[i]  mod 2 <> 0
    alors
       X ← vect[i]              % Permutation des éléments vect[i] et vect[j] %
       vect[i] ← vect[j]
       vect[j] ←X
      j ← j-1
    sinon
      i ←i+1
    fsi
  jusqu'à  i = j
  % Affichage %
  écrire (' Les éléments du vecteur Vect après rangement sont : ')
  pour i de 1 à N
   faire
    écrire ('Vect(',i,') = ',vect[i])
   ffaire
Fin
```

6.

Nous procédons d'abord à la saisie du vecteur de noms (*vect_chain*) rangés suivant l'ordre lexicographique. Pour cela, nous introduisons le premier nom *(vect_chain[1])*, puis nous utilisons une boucle *Pour* pour saisir les autres noms. A chaque itération, nous nous assurons que le nom lu est supérieur ou égal au nom lu précédemment (exprimé par la condition *vect_chain[I] >= vect_chain[I-1])*.

En ce qui concerne la recherche d'un nom, donné dans le vecteur trié *vect_chain*, nous adoptons le principe de dichotomie (défini au paragraphe 1.1.4 du chapitre V) et qui consiste à :

1. Initialiser *D* (borne inférieure de l'intervalle de recherche) à 1 et *F* (borne
   supérieure de l'intervalle de recherche) à 100.
2. Diviser l'intervalle en deux pour obtenir *M* l'indice de l'élément du milieu.
3. Comparer l'élément d'indice *M (vect_chain [M])* et la valeur recherchée *val_chain* :
   - Si vect_chain[M] > val_chain nous limitons la recherche dans l'intervalle [D, M-1].
   - Si vect_chain[M] < val_chain nous limitons la recherche dans l'intervalle [M+1, F] .

Nous répétons les deux opérations 2 et 3 jusqu'à trouver la valeur recherchée (exprimé par la condition *vect_chain[M] = val_chain*) ou jusqu'à ce que l'intervalle de recherche soit vide (exprimé par la condition *D>F*).

```
Algorithme Rech_dicho

 Variable
   vect_chain : tableau [1..100] de chaine
   val_chain : chaine
   I : 1..100
   M, D, F : 0..100

 Début
   écrire ('Application du principe de la dichotomie sur un tableau de noms')
   écrire ('Saisie des éléments du vecteur en le classant par ordre croissant ')
   lire (vect_chain[1])
   pour I de 2 à 100
    faire
     répéter
      lire (vect_chain[I])
     jusqu'à vect_chain[I] >= vect_chain[I-1]
    ffaire
   écrire ('Saisie du nom recherché ')
   lire (val_chain)
```

```
% Recherche dichotomique %
D ← 1
F ← 100
répéter
  M ← (D+F) div 2
  si vect_chain[M] < val_chain
  alors
    D ← M+1
  sinon
    si vect_chain[M] > val_chain
    alors
      F ← M-1
    fsi
  fsi
jusqu'à (vect_chain[M] = val_chain) ou (D > F)
si vect_chain[M] = val_chain
alors
  écrire ( val_chain,' se trouve dans le vecteur')
sinon
  écrire (val_chain,' ne se trouve pas dans le vecteur')
fsi
Fin
```

7.

Nous utilisons la variable *P* (initialisée à 1) pour récupérer le produit des éléments de la matrice *Mat*.
Ensuite nous parcourons cette matrice ligne par ligne en nous servant de 2 boucles *Pour* imbriquées (la première boucle externe pour avancer dans les lignes, et la deuxième interne pour avancer dans les colonnes) pour :

- Saisir l'éléments *Mat[I,J]* de la matrice , *I* représente le numéro de la ligne et *J* représente le numéro de la colonne.
- Effectuer le produit de *P* par l'élément *Mat[I,J]* et mettre le résultat dans *P*.

NB : Le parcours de la matrice peut se faire colonne par colonne ; il suffit de permuter les deux boucles.

```
Algorithme Pro_mat

 Variable
  Mat : Tableau[1..200,1..300] de réel   % Le nombre maximal des lignes
                                            est 200 et le nombre maximal
                                            des colonnes est 100 %
  I, n : 1..200
  J, m : 1..300
  P : réel

 Début
  écrire (' Produit des éléments d''une matrice à valeurs réelles')
  % Saisie et calcul du produit %
  écrire ('Introduire la taille de la matrice')
  lire (n,m)
  écrire ('saisir les éléments de la matrice de ',n,' lignes et ',m,' colonnes et
          effectuer leur produit')
  P ← 1
  pour I de 1 à n
   faire
    pour J de 1 à m
     faire
      lire (Mat[I, J])
      P ← P*Mat[I, J]
     ffaire
   ffaire
  % Affichage %
  écrire ('Le produit des éléments de la matrice est égal à : ',P)
 Fin
```

8.

Une matrice carrée est une matrice dont le nombre de lignes est égal au nombre de colonnes. Pour mettre à zéro les éléments des deux diagonales, il suffit de constater que les éléments de la *première* diagonale (diagonale principale) sont caractérisés par l'égalité entre le numéro de la ligne et le numéro de la colonne, et que les éléments de la deuxième diagonale sont définis par l'expression : numéro de la colonne = nombre de lignes-numéro de ligne +1 (le nombre de lignes est fixé à 50).

Nous ne manipulerons donc, qu'un seul indice vu la relation existant entre le numéro de la ligne et le numéro de la colonne.
NB : Nous supposons que la matrice est saisie.

```
Algorithme Diag_Zero

 Variable
   M : Tableau[1..50,1..50] de  réel
   i, j: 1..50

 Début
   écrire ('Mettre à zéro les 2 diagonales d''une matrice carrée')
   pour i de 1 à 50
    faire
     M[i, i] ← 0              %Mettre à 0 les éléments de la 1ere diagonale %
     M[i, 50-i+1] ← 0       % Mettre à 0 les éléments de la 2ieme diagonale %
    ffaire
   % Affichage %
   pour i de 1 à 50
    faire
     pour j de 1 à 50
      faire
       écrire ('M(', i, ',' ,j , ') = ', M[i, j])
      ffaire
    ffaire
 Fin
```

9.

Pour parcourir la partie de la matrice située au-dessus de la diagonale principale (ligne par ligne), nous remarquerons que si la ligne considérée est *I*, les colonnes *J* parcourues doivent varier de *I+1* à *100*. *I* prend les valeurs de 1 à 99.
A chaque itération, si la valeur de l'élément *M[I,J]* est comprise entre 10 et 30 , nous ajoutons 1 au compteur *N* (initialisé à 0) et à la somme *Som* (initialisée à 0) la valeur de l'élément *M[I,J]*.
N contient le nombre des éléments *M[I,J]* situés au-dessus de la diagonale compris entre 10 et 30, et *Som* contient leur somme.
NB : Les éléments de la matrice *M* sont supposés saisis.

```
Algorithme Dessus_diag

 Variable
  M : tableau[1..100,1..100] de 0..max_entier
  I, J : 1..100
  N, Som : 0..max_entier

 Début
  écrire ('Déterminer le nombre d''éléments au-dessus de la diagonale
         compris entre 10 et 30 et faire leur somme ')
  % Traitement %
  N ← 0
  Som ← 0
  pour I de 1 à 99
   faire
    pour J de I+1 à 100
     faire
      si  (M[I, J]>10) et (M[I, J]<30)
      alors
       N ← N+1
       Som ← Som + M[I, J]
      fsi
     ffaire
   ffaire
  % Affichage %
  écrire (' Le nombre d''éléments au-dessus de la diagonale dont la valeur
         est comprise entre 10 et 30 est égal à ', N, ' et leur somme est
         égale à ', Som )
Fin
```

10.

Nous supposons, initialement, que la première ligne constitue la ligne dont la somme des éléments est maximale. Cette somme (*som*) est sauvegardée dans la variable *max*, et 1 est mémorisé dans la variable *ligmax* (représentant l'indice de ligne dont la somme des éléments est maximale).

Pour chacune des lignes restantes, de 2 à 200, nous effectuons les opérations suivantes :

- Initialiser la somme *som* à 0.
- Calculer la somme de la ligne considérée.
- Comparer la nouvelle somme *som* à *max* ; si *som* est supérieure à *max* nous stockons *som* dans *max* et nous sauvegardons le numéro de la ligne dans la variable *ligmax*.

```
Algorithme Lign_somax

 Variable
  M : Tableau[1..200,1..100] de entier
  i, ligmax : 1..200
  j : 1..100
  som, max : entier

 Début
  écrire (' Déterminer la ligne dont la somme des éléments est maximale')
  som ← 0
  pour j de 1 à 100
   faire
    som ← som+ M[1, j]
   ffaire
  max ← som            % Initialiser Max à la somme de la première ligne %
  ligmax ← 1
  pour i de 2 à 200
   faire
    som ← 0              % Pour chaque ligne, réinitialiser Som à 0 %
    pour j de 1 à 100
     faire
       som ← som+ M[i, j]
     ffaire
    si som > max       % Après le calcul de la somme de chaque ligne,
    alors                 comparer cette somme à max %
     max ← som
     ligmax ← i
    fsi
   ffaire
  % Affichage %
  écrire ('La ligne dont la somme de ses éléments est maximale est : ',
         ligmax)
Fin
```

11.

Il s'agit dans cet exercice de décrire la feuille de soins médicaux comme un enregistrement composé de champs représentant les renseignements d'un assuré.

```
Type
 Date = Article
          J : 1..31
          M : 1..12
          A : 1000..3000
        Fin

  Fich_med = Article
                nom, prenom : chaine
                date_nais : Date
                lieu_nais : chaine
                adr : chaine
                nom_emp : chaine
                adr_emp : chaine
                mod_paie : chaine
             Fin
```

12.

Un nombre complexe est défini comme un enregistrement composé de deux champs représentant respectivement la partie réelle et la partie imaginaire.

```
Algorithme complexe

 Type
  Complex : Article
              preel : réel
              pimag : réel
            Fin

 Variable
  C1, C2, C3 : Complex
```

a.

Pour calculer le produit de complexes, nous saisissons d'abord les deux nombres complexes c'est à dire, la partie réelle et la partie

imaginaire de chaque nombre complexe et nous effectuons le produit de ces deux nombres complexes en utilisant la formule suivante :
(X1+iY1)*(X2+iY2)=(X1*X2-Y1*Y2)+i(X1*Y2+X2*Y1).

```
Début
  écrire ('Produit de deux complexes')
  % Saisie %
  ecrire (' Saisir le 1er complexe')
  lire (C1.preel, C1.pimag)          %C1 et C2 sont les données  et C3 la
                                       somme de C1 et C2 %
  écrire (' Saisir le 2eme complexe')
  lire(C2.preel, C2.pimag)
  % Produit %
  C3.preel ← C1.preel*C2.preel - C1.pimag*C2.pimag
  C3.pimag ← C1.preel*C2.pimag + C1.pimag*C2.preel
  % Affichage %
  écrire ('(',C1.preel,' +i ',C1.pimag,')*(',C2.preel,' +i ',C2.pimag,') =',
         C3.preel, ' +i ', C3.pimag)
Fin
```

b.

Nous ne pouvons comparer deux enregistrements globalement. Pour comparer les deux nombres complexes nous procédons de la manière suivante :

- Si la partie réelle du premier nombre complexe est égale à la partie réelle du deuxième nombre complexe, nous passons à la comparaison de la partie imaginaire des deux nombres complexes : si ces deux parties sont égales nous déduisons que les nombres complexes sont égaux.
- Si la partie réelle du premier nombre complexe est différente de la partie réelle du deuxième nombre complexe, nous déduisons que les nombres complexes ne sont pas égaux (sans comparer les parties imaginaires).

```
Début
  écrire('comparaison des 2 complexes C1 et C2 ')
  %On suppose que les 2 complexes C1 et C2 sont saisis %
  si C1.preel = C2.preel
  alors
    si C1.pimag = C2.pimag
```

```
  alors
    écrire ('Les deux nombres complexes sont égaux')
  sinon
    écrire ('Les deux nombres complexes ne sont pas égaux')
  fsi
 sinon
  écrire ('Les deux nombres complexes ne sont pas égaux')
 fsi
Fin
```

13.

Le vecteur *fich* est défini comme un tableau dont les éléments sont de type enregistrement composé du code, de l'auteur, du titre, de l'éditeur et de l'année d'édition.
Pour la saisie, nous introduisons les valeurs des différents champs de chaque élément du vecteur fich. Concernant l'affichage de tous les auteurs ayant édité chez Chihab en l'année 2001, nous parcourons le vecteur *fich*, et à chaque itération, nous testons le champ *edit* et le champ *annee* : s'ils correspondent respectivement à *Chihab* et à *2001*, nous affichons le nom et le prénom de l'auteur.

```
Algorithme list_auteur

 Type
  Inf = Article
         code : chaine
         auteur : Article
                   nom, prenom : chaine
                  Fin
        titre : chaine
        edit : chaine
        annee : 1000..3000
       Fin

 Variable
  fich : tableau [1..1000] de Inf
  i : 1..1000
```

```
Début
 écrire ('Saisir les informations')
 pour i de 1 à 1000
  faire
   lire (fich[i].code, fich[i].auteur.nom, fich[i].auteur.prenom, fich[i].titre,
        fich[i].edit, fich[i].annee)
  ffaire
 écrire ('Liste des auteurs ayant édité en 2001 chez Chihab')
 pour i de 1 à 1000
  faire
   si (fich[i].edit = 'Chihab') et (fich[i].annee = 2001)
   alors
    écrire (fich[i].auteur.nom, fich[i].auteur.prenom)
   fsi
  ffaire
Fin
```

14.

Nous introduisons la phrase caractère par caractère et nous testons si le caractère lu appartient à l'ensemble [., „ ;, !, :, ?]. Dans l'affirmative, nous incrémentons le compteur *compt* (initialisé à 0) par pas de 1. Nous répétons ce processus jusqu'à atteindre la fin de la phrase indiquée par le caractère point.

```
Algorithme compt_ponct
 Variable
   c : caractère
   compt : 0..max_entier
 Début
  écrire ('Introduire la phrase caractère/caractère')
  compt ← 0
  répéter
     lire (c)
     si c dans ['.', ',', ';', '?', ':', '!']
     alors
      compt ← compt +1
     fsi
  jusqu'à c = '.'
 écrire ('Le nombre de caractères de ponctuations dans cette phrase est égal
         à ',compt)
 Fin
```

15.

Pour construire l'ensemble des caractères alphabétiques minuscules (Alpha) qui n'appartiennent pas dans la phrase, nous procédons de la manière suivante :

- Nous initialisons *Alpha* à l'ensemble constitué de tous les caractères alphabétiques minuscules ([‘a’..’z’]) .
- La phrase est introduite caractère par caractère, si le caractère lu est un caractère alphabétique minuscule, c'est à dire appartient à *Alpha*, nous le retranchons de *Alpha* : cette opération consiste à effectuer la différence entre l'ensemble *Alpha* et l'ensemble constitué du caractère lu (ce dernier est mis entre crochets). Nous répétons ce processus jusqu'à atteindre la fin de la phrase indiquée par le point.
- Pour afficher les éléments de l'ensemble *Alpha* : nous testons si chaque compteur *i*, variant du caractère *a* au caractère *z*, appartient à *Alpha*. Si c'est le cas, nous affichons *i*.

```
Algorithme Ens_alphmin

 Variable
  Alpha : ensemble de ‘a’ ..’z’
  C : caractère
  i : ‘a’..’z’

 Début
  écrire (‘Ensemble des caractères alphabétiques minuscules n’’apparaissant
          pas dans la phrase’)
  % Construction de l’ensemble Alpha %
  Alpha ← [‘a’..’z’]
  répéter
    lire (C)
    si C dans Alpha
    alors
      Alpha ← Alpha - [C]
    fsi
  jusqu’à  C =’.’
  % Affichage %
  écrire (‘Les éléments de l’’ensemble sont { ‘)
```

```
  pour i de 'a' à 'z'
   faire
    si i dans Alpha
    alors
     écrire (i,' ')
    fsi
   ffaire
  écrire ('}')
 Fin
```

16.

Pour construire l'ensemble des chiffres caractères (chif) apparaissant dans la phrase, nous effectuons les opérations suivantes :

- Initialiser l'ensemble *chif* à l'ensemble vide.
- Saisir la phrase caractère/caractère, si le caractère introduit est un caractère chiffre c-à-d appartient à l'ensemble ['0'..'9'], l'ajouter à l'ensemble *chif* : cette opération consiste à effectuer l'union entre l'ensembles *chif* et l'ensemble constitué du caractère lu. Répéter ce processus jusqu'à atteindre la fin de la phrase.
- Pour afficher les éléments de l'ensemble *chif* : tester si chaque valeur de *I*, variant de '0' à '9', appartient à *chif*. Si c'est le cas, afficher *I*.

```
Algorithme Ens_chiffr
 Variable
  c : caractère
  chif : ensemble de '0'..'9'
  I : '0'..'9'
 Début
  écrire ('La phrase est introduite caractère/caractère')
   % Construction de l'ensemble chif %
   chif ← []
  répéter
    lire (c)
    si c dans ['0'..'9']
    alors
      chif ← chif + [c]
    fsi
  jusqu'à  c = '.'
```

```
% Affichage %
écrire ('Les éléments de l''ensemble  sont {')
pour I de '0' à '9'
 faire
  si I dans chif
  alors
   écrire (I, ' ')
  fsi
 ffaire
écrire ('}')
Fin
```

Corrigé des exercices du chapitre VI

1.

Nous montrons à travers cet exercice, un exemple de fonction qui fait appel à une autre fonction. Il s'agit de la fonction *min3* qui doit fournir le minimum de 3 nombres *x*, *y* et *z* en appelant la fonction *min2*, deux fois. Le premier appel calcule le minimum des 2 premiers *nombres x et y et, le deuxième appel détermine* le minimum entre le résultat obtenu après le premier appel et le troisième nombre z.
La fonction *min2* compare deux nombres et retourne le plus petit parmi eux.

```
fonction min2 (E x, y : réel) : réel          % min2 : fonction qui calcule le
                                                minimum de deux nombres %
 Début
  si x < y
  alors
   min2 ← x    % Instruction de retour du résultat %
  sinon
   min2 ← y    % Instruction de retour du résultat %
  fsi
Fin

fonction min3 (E x, y, z : réel) : réel    % min3 : fonction qui calcule le mini-
                                             mum de 3 nombres en utilisant min2 %
 fonction
  min2 : réel                 % Déclaration de la fonction min2 dans la partie
                                déclaration de la fonction min3 %
 Début
  min3 ← min2 ( min2 (x, y), z)    % Appel de la fonction min2 (deux fois)
                                     et retour du résultat %
 Fin
```

2.

Pour le calcul de $C_n^p = n!/\ p!(n-p)!$, nous appelons trois fois la fonction *fact*, pour calculer n!, p! et (n-p)! respectivement.
Concernant l'algorithme de la fonction *fact*, nous rappelons que
N! = 1.2.3…(N-1).N. Cela revient à utiliser :

- une variable (représentée par *F* dans l'algorithme) dans laquelle nous effectuons le produit et que nous initialisons à 1, et
- une boucle Pour avec un compteur (I) variant de 1 à *N*.

A chaque itération, nous multiplions *F* par *I* et sauvegardons le résultat dans *F*. La valeur de *F* obtenue à la sortie de la boucle Pour constitue le factoriel du nombre *N* donné.

```
Algorithme calcul_cnp

 Variable
  N, P : 0.. max_entier

 fonction
  fact : 1..max_entier

 Début
  écrire ('Calcul de CNP ')
  écrire ('Introduire les données')
  lire (N, P)
  écrire ('C( ', N, ', ', P, ' ) = ', fact(N) div (fact(P) * fact(N-P))
 Fin

fonction fact (E N : 0.. max_entier): 1.. max_entier   % fact : fonction qui
                                                          calcule le factoriel
 Variable                                                 d'un nombre %
   I, F : 1.. max_entier

 Début
  F ← 1
  pour I de 1 à N
   faire
    F ← F* I
   ffaire
  fact ← F              % Instruction de retour du résultat %
 Fin
```

3.

Pour calculer le PPCM de 2 nombres entiers naturels, nous appliquons la formule :

$$PPCM(A, B) = \frac{A * B}{PGCD(A, B)}$$

qui fait appel à une fonction qui détermine le plus grand commun diviseur (*PGCD*) de deux nombres.

La méthode utilisée pour le calcul du PGCD de deux nombres entiers naturels est celle proposée par l'énoncé :

PGCD(A, 0) = A (1)

PGCD(A, B) = PGCD(B, A mod B) (2)

Exemple:

PGCD(24, 18) = PGCD(18, 6) = PGCD(6, 0) en appliquant (2)

PGCD(6, 0) = 6 en appliquant (1) d'où PGCD(24, 18) = 6.

En remplaçant A par x et B par y dans les égalités (1) et (2), nous obtenons l'algorithme de la fonction PGCD donné ci-dessous.

```
Algorithme PPCM

 Variable
  A, B : 0.. max_entier

 Fonction
  PGCD: 1.. max_entier

 Début
  écrire ('Calcul du plus petit multiple commun de deux nombres entiers
          naturels')
  écrire ('Introduire les deux nombres entiers naturels')
  lire (A, B)
  écrire ('PPCM( ', A, ', ', B, ') = ', (A*B) div PGCD(A, B))
 Fin

Fonction PGCD (E x, y : 0.. max_entier) : 1.. max_entier

 Variable
  z : 0.. max_entier          % Variable de sauvegarde %
```

```
Début
  tantque y ≠ 0
   faire
    z ← x
    x ← y
    y ← z mod y
   ffaire
  PGCD ← x
Fin
```

4.

Pour afficher tous les nombres entiers d'Armstrong inférieurs à un *nombre donné N, nous faisons* appel *à la* fonction *arms* qui vérifie si un nombre entier naturel est un nombre d'Armstrong.

Un nombre d'Armstrong est égal à la somme des cubes de ses chiffres. Pour obtenir les chiffres composant un nombre entier, nous adoptons le principe des divisions (entières) successives par *10* à ce nombre. Nous arrêtons les divisions lorsque nous obtenons un quotient nul. Les restes de ces divisions constituent les différents chiffres composant ce nombre. Par exemple, pour trouver les chiffres composant le nombre 125 nous effectuons les opérations suivantes :

- 125 div 10 donne un quotient Q = 12 et un reste R = 5.
- 12 div 10 donne un quotient Q = 1 et un reste R = 2.
- 1 div 10 donne un quotient Q = 0 et un reste R = 1.

Nous arrêtons le processus car nous avons obtenu un quotient nul (*Q=0*) et les chiffres composant le nombre 125 sont donnés par les différents restes obtenus à chaque étape et qui sont : 5, 2 et 1.

Vérifier qu'un nombre donné *M* est un nombre d'Armstrong revient à :

- calculer le cube de chacun des chiffres qui le composent en utilisant une fonction (*cube*),
- l'ajouter à une somme (*som*) initialisée à 0,
- comparer *M* à la somme des cubes de tous les chiffres qui le composent.

```
Algorithme nbre_arms

 Variable
  N : 0.. max_entier
  I : 0.. max_entier

 Fonction
  arms : booléen

 Début
  écrire ('Les nombres d''Armstrong inférieurs à un entier naturel donné
        N')
  écrire ('Introduire l''entier naturel N')
  lire (N)
  écrire ('La liste des nombres d''Armstrong inférieurs à l''entier naturel ',
          N, ' est la suivante :')
  pour I de 0 à N-1
   faire
    si arms(I)
    alors
     écrire (I, ' ')
    fsi
   ffaire
 Fin

Fonction cube (E A : 0.. max_entier): 0.. max_entier    % cube : fonction
                                                          calculant le cube
 Début                                                    d'un nombre %
   cube ← A*A*A
 Fin

Fonction arms (E NB : 0.. max_entier): booléen    % arms vérifie si un
                                                    nombre donné est un
 Variable                                           nombre d'Armstrong%
  som, M : 0.. max_entier

 Fonction
  cube : 0.. max_entier
```

```
Début
  M ← NB          % Sauvegarde du nombre NB dans M %
  som ← 0
  répéter
    som ← som + cube(NB mod 10)  % arms est un exemple de fonction qui
                                    fait appel à une autre fonction (cube)%
    NB ← NB div 10
  jusqu'à  NB = 0
  si som = M          % Comparaison de la somme des cubes des chiffres
  alors                 composant M à M %
    arms ← vrai
  sinon
    arms ← faux
  fsi
Fin
```

5.

Pour effectuer le classement de trois nombres entiers *A*, *B* et *C*, nous appelons trois fois la procédure *ECHANGE* de deux nombres entiers vue au paragraphe 4 du chapitre VI.

- Le premier appel effectue l'échange entre les nombres *A* et *B* (la plus petite valeur est retenue dans *A*).
- Le deuxième appel effectue l'échange entre les nombres *B* et *C* (la plus petite valeur est retenue dans *B*).
- Le troisième appel effectue l'échange entre les nombres *A* et *B* (la plus petite valeur est retenue dans *A*).

Exemple

	A	B	C
	4	2	1
1er appel	2	4	1
2ème appel	2	1	4
3ème appel	1	2	4

Algorithme classement

Variable
  A, B, C : entier

Procédure
  ECHANGE

Début
  écrire ('Classement de trois nombres entiers par ordre croissant')
  écrire ('Introduire trois nombres entiers')
  lire (A, B, C)
  ECHANGE (A, B)
  ECHANGE (B, C)
  ECHANGE (A, B)
  écrire ('Les nombres classés par ordre croissant sont : ', A, ', ', B, ', ', C)
Fin

6.

Nous initialisons la variable *min* à la valeur de l'élément *M[1, 1]* et sauvegardons l'indice de ligne (1) dans *RL* et l'indice de colonne (1) dans *RC*. Nous procédons ensuite, au parcours de la matrice ligne par ligne en utilisant 2 boucles Pour imbriquées. A chaque itération, si la valeur de l'élément *M[I, J]* est inférieure à *min*, nous mémorisons la valeur de l'élément *M[I, J]* dans *min*, le numéro de ligne *I* dans *RL*, et le numéro de colonne *J* dans *RC*.

La matrice constitue un paramètre d'entrée pour la procédure, et RL et RC sont des paramètres de sortie (valeurs récupérables par l'algorithme appelant).

a.

Type
  MAT = tableau[1.. 100, 1.. 200] de réel

b.

Procédure position_min (E M : MAT, S RL : 1.. 100, S RC : 1.. 200)
  Variable
    I : 1.. 100
    J : 1.. 200
    min: réel   %min représente l'élément minimum de la matrice %

```
Début
  min ← M[1, 1]
  RL ← 1                 % RL représente l'indice de ligne de l'élément
                            min de la matrice %
  RC ← 1                 % RC représente l'indice de la colonne de
                            l'élément min de la matrice %
  pour I de 1 à 100
   faire
    pour J de 1 à 200
     faire
      si M[I, J] < min
      alors
        min ← M[I, J]
        RL ← I
        RC ← J
      fsi
     ffaire
   ffaire
Fin
```

7.

Pour la procédure de saisie (*Saisie_mat*) de la matrice *M* d'assiduité des étudiants nous devons s'assurer que la valeur de l'élément introduit est soit la caractère A (Absent) ou le caractère P (Présent).

Comme la fonction *compte_abs* détermine le nombre d'absence d'un étudiant de numéro donné, elle a comme paramètres d'entrée la matrice *MAT* et le numéro de l'étudiant *n*. La tâche de cette fonction est de parcourir la ligne *n* et d'incrémenter le compteur des absences (*c*) de l'étudiant *n*, d'un pas de 1 à chaque fois que nous rencontrons un élément dont la valeur est égale au caractère A.

Pour afficher le nombre d'absences de chaque étudiant au niveau de l'algorithme appelant (*Assiduite*), nous faisons appel à la procédure *Saisie_mat* pour saisir la matrice *M*, et à la fonction *compte_abs* pour calculer le nombre d'absences de chacun des étudiants numérotés de 1 à 30.

```
Algorithme Assiduite

 Type
  PRESENCE = tableau[1.. 30, 1.. 20] de caractère
 Variable
     M : PRESENCE
     I : 1.. 30
 Procédure
     Saisie_mat
 Fonction
     compte_abs : 0 ..20

 Début
  Saisie_mat(M)                    % Appel de la procédure Saisie_mat %
  pour I de 1 à 30
   faire
    % Appel à la fonction compte_abs dans une expression %
    écrire ('Le nombre d''absences de l''étudiant ', I, ' est : ',
            compte_abs(M, I))
   ffaire
 Fin

Procédure Saisie_mat (S MAT: PRESENCE)      % La saisie de la matrice
                                              représentant l'assiduité
 Variable                                      des étudiants %
  I : 1.. 30
  J: 1.. 20

 Début
  pour I de 1 à 30
   faire
    pour J de 1 à 20
     faire
      répéter
        écrire ('Saisir la valeur P si l''étudiant est présent et la valeur A s'il
                est absent')
        lire (MAT[I, J])
      jusqu'à (MAT[I, J] = 'A') ou (MAT[I, J] = 'P')
     ffaire
   ffaire
 Fin
```

Fonction compte_abs (E MAT : PRESENCE, E n : 1.. 30) : 0.. 20

```
Variable
 I : 1.. 20
 c : 0.. 20

Début
 c ← 0
 pour I de 1 à 20
  faire
   si MAT[n, I] = 'A'
   alors
    c ← c + 1
   fsi
  ffaire
 compte_abs ← c
Fin
```

8.

Les éléments du vecteur du type *RESULTAT* sont de type enregistrement composé de 2 champs : *note* et *nom* du candidat. La procédure *saisie* permet de saisir le nom et la note de chacun des 200 candidats en s'assurant que la note est comprise entre 0 et 20. Pour la fonction *moy*, nous accédons au champ *note* de chacun des 200 éléments du vecteur *V* et nous calculons la moyenne des 200 notes.
En ce qui concerne l'algorithme appelant *liste_candidats*, nous faisons appel à la procédure *saisie* pour introduire les informations des 200 candidats, et nous comparons la note de chacun des 200 candidats à la moyenne de l'examen *M* calculée par la fonction *moy*. Ensuite, nous affichons le nom des candidats dont la note est supérieure ou égale à cette moyenne.

```
Algorithme liste_candidats

 Type
  res = Article
          nom : chaîne
          note : réel
        Fin
  RESULTAT = tableau[1..200] de res
```

Variable
 T : RESULTAT
 M : réel
 I : 1..200

Procédure
 Saisie

Fonction
 moy : réel

Début
 écrire ('Affichage de la liste des candidats dont la note est supérieure à la moyenne d''un examen')
 saisie(T)
 M ← moy(T)   % M représente la moyenne de l'examen %
 écrire ('La liste des candidats dont la note est supérieure à la moyenne ', M, ' est : ')
 pour I de 1 à 200
  faire
   si T[I].note >= M
   alors
    écrire (T[I].nom, ' ')
   fsi
  ffaire
Fin

Procédure saisie (S V : RESULTAT)   % La saisie du nom des candidats et de leur note d'examen %

Variable
 I : 1..200

Début
 pour I de 1 à 200
  faire
   écrire ('Saisir le nom du candidat')
   lire (V[I].nom)

```
    répéter
      écrire ('Saisir la note du candidat')
      lire (V[I].note)
    jusqu'à (0<=V[I].note) et (20>=V[I].note)   %vérifier que 0≤ note≤ 20%
  ffaire
Fin
```

```
Fonction moy (E V : RESULTAT): réel        % moy : fonction qui calcule la
                                             moyenne de l'examen %
 Variable
  I : 1..200
  som: reel

 Début
  som ← 0
  pour I de 1 à 200
   faire
    som ← som + V[I].note
   ffaire
  moy ← som/200
 Fin
```

9.

Un nombre rationnel est donné sous la forme de P/Q. Il peut être représenté par un enregistrement qui se compose de deux champs : numérateur et dénominateur. Pour saisir un nombre rationnel il faut éviter un dénominateur nul. La somme de deux rationnels peut fournir un résultat non simplifié ; pour réduire ce résultat, nous utilisons la fonction PGCD vue dans la solution de l'exercice 3. Diviser le numérateur et le dénominateur du résultat, par leur PGCD revient à simplifier ce résultat.

```
Algorithme somme_rat

 Type
  rat = Article
         num : entier     % le numérateur d'un nombre rationnel %
         den : entier     % le dénominateur d'un nombre rationnel %
      Fin
```

```
% Affichage de l'ensemble des éléments en dessous de la diagonale
   principale %
écrire ('Les éléments de cet ensemble sont : {')
pour c de 0 à 9
 faire
   si c dans E1
   alors
     écrire (c, ' ')
   fsi
  ffaire
 écrire (' }')
Fin
```

11.

Pour déterminer le nombre d'apparition de chaque lettre alphabétique dans la phrase, nous définissons un vecteur dont l'intervalle d'indiçage est 'a'..'z' et dont le type des éléments est entier naturel. Pour la construction de ce vecteur, nous utilisons la procédure *construire_vect*, ayant comme paramètre de sortie le vecteur *V* et, dont la tâche est de :

- Initialiser les éléments du vecteur *V* à 0
- Introduire la phrase caractère par caractère, si le caractère lu (*c*) est une lettre alphabétique (c-à-d appatenant à ['a'..'z'] , incrémenter l'élément de *V*, indicé par *c*, par un pas de 1. Répéter ce processus jusqu'à atteindre la fin de la phrase.

Pour construire l'ensemble, contenant les lettres alphabétiques apparaissant dans la phrase, à partir du vecteur conçu, nous définissons la procédure *construire_ens* ayant comme paramètre d'entrée un vecteur (*T*) et comme paramètre de sortie l'ensemble en question (*E3*), et dont la tâche est de :

- Initialiser l'ensemble *E3* à vide.
- Parcourir le vecteur *T*, si la valeur de l'élément de *T* est différente de 0, faire l'union entre l'ensemble *E3* et l'ensemble constitué de l'indice de l'élément du vecteur *T*.

La procédure *affichage_ens* permet d'afficher les éléments d'un ensemble donné et est définie de la même manière que dans le bloc de la solution de l'exercice 15 du chapitre V.

L'algorithme appelant (*phrase_vecteur*) fait appel à

- la procédure *construire_vect* deux fois car nous disposons de deux phrases,
- la procédure *construire_ens* deux fois pour construire les deux ensembles E1 et E2 qui contiennent, respectivement, les lettres alphabétiques de la 1ère phrase et de la 2ème phrase,
- la procédure *affichage_ens* une première fois pour afficher l'ensemble des lettres alphabétiques communes aux 2 phrases (noté *E1*E2* : intersection entre E1 et E2), et une seconde fois pour afficher l'ensemble des lettres alphabétiques figurant dans la 1ère phrase et ne figurant pas dans la 2ème phrase (noté *E1-E2* : différence de deux ensemble).

```
Algorithme phrase_vecteur

 Type
  vect = tableau['a'.. 'z'] de 0.. max_entier
  ens = ensemble de 'a'.. 'z'

 Variable
  V1, V2 : vect
  E1, E2 : ens

 Procédure
  construire_vect
  construire_ens
  affichage_ens

 Début
  construire_vect(V1)      % Construction du 1er vecteur V1 à partir de la
                              première phrase saisie %
  construire_vect(V2)      % Construction du 2ème vecteur V2 à partir de la
                              deuxième phrase saisie %
  construire_ens (V1, E1)    % Construction du premier ensemble E1 à
                                partir de premier vecteur V1 %
  construire_ens (V2, E2)    % Construction du deuxième ensemble E2 à
                                partir du deuxième vecteur V2 %
```

```
   affichage_ens (E1* E2)      % Affichage des éléments communs entre E1
                                 et E2 %
   affichage_ens (E1- E2)      % Affichage des éléments appartenant à E1 et
                                 n'appartenant pas à E2 %
  Fin

procédure construire_vect (S V :vect)   % Procédure qui construit un vecteur
  Variable                                 à partir d'une phrase %
   c : caractère
   i : 'a'.. 'z'

  Début
   pour i de 'a' à 'z'
    faire
      V[i] ← 0
    ffaire
   écrire ('Saisir une phrase se terminant par un point, caractère par
           caractère')
   répéter
    lire (c)
    si c dans ['a'.. 'z']
    alors
      V[c] ← V[c] + 1
     fsi
  jusqu'à  c = '.'
 Fin

procédure construire_ens (E T: vect, S E3 : ens)   % Procédure qui construit
 Variable                                             un ensemble à partir
  i : 'a'..'z'                                        d'un vecteur %

 Début
 E3 ← []
 pour i de 'a' à 'z'
   faire
    si T[i] ≠ 0
    alors
     E3 ← E3 + [i]
    fsi
   ffaire
 Fin
```

```
procédure affichage_ens (E E4 : ens)        % Procédure qui affiche les
                                               éléments d'un ensemble %
 Variable
  i : 'a'..'z'

 Début
  écrire ('Les éléments de cet ensemble sont : { ')
  pour i de 'a' à 'z'
   faire
    si i dans E4
    alors

     écrire (i, ' ')
    fsi
   ffaire
  écrire (' }')
 Fin
```

BIBLIOGRAPHIE

[1] Joëlle Biondi et Gille Clavel. Introduction à la programmation : Algorithmique et langages. Tome 1, eds. Masson, 1987.

[2] E. A. Chambers. Notions élémentaires d'informatique. Eds. Guerin, Montréal (Québec) 1988.

[3] Guy Chaty et Jean Vicard, Programmation : cours et exercices. Ellypses, Eds. MARKETING, Paris, 1992.

[4] Claude Delannoy. Initiation à la programmation. Eds. Eyrolles, Paris, 1986.

[5] Marc Ducamp et Noël Millet. Cours d'informatique. Eds. Eyrolles, Paris, 1988.

[6] Grégoire (nom collectif : G. Benay, V. Donzeau Gouge, H. Thanh Té, C. Kaiser, P. Lignelet, A. M. Rasser, F. Y. Villemin). Informatique-Programmation : La programmation structurée. Tome 1, eds. Masson, 1986.

[7] Jacques Guyot et Christian Vial. Arbres, tables et algorithmes. Eds. Chihab 1994, eds. Eyrolles 1989-1992.

[8] Jean-Pierre Laurent et Jacqueline Ayel. Exercices commentés d'analyse et de programmation. Bordas, Paris, 1985.

[9] Gérard Leblanc. Borland C++ Builder. Eds. Eyrolles, Paris, 2000.

[10] Dominique Maille, Programmation en langage Pascal : du PASCAL standard à turbo-PASCAL. Eds. Berti, Alger, 1992.

[11] Jean-Pierre Meinadier. Structure et fonctionnement des ordinateurs. Paris, Librairie Larousse, 1988.

[12] B. Meyer, C. Baudoin. Méthodes de programmation. Eds. Eyrolles, collection DER/EDF, 1978.

[13] Jean-Louis Nebut. Théorie et pratique du langage Pa[illegible] Technip., Paris, 1980

[14] Chantal Richard et Patrice Richard. Initiation à l'algorithmique [illegible] 135 exercices corrigés. Eds. Belin, Paris, 1985

[15] Alfred Strohneir. Le matériel informatique : Concepts et principes, Lausane. Presses polytech. Romandes, [illegible]

Achevé d'imprimer sur les presses de

l'OFFICE DES PUBLICATIONS UNIVERSITAIRES

1, Place Centrale - Ben-Aknoun - ALGER